Journal of Machine Learning Research 22 (2021) 1-38 Submitted 7/19; Revised 11/20; Published 1/21

A Two-Level Decomposition Framework Exploiting First and Second Order Information for SVM Training Problems

Giulio Galvan GIULIO.GALVAN@UNIFI.IT
Dipartimento di Ingegneria dell'Informazione
Università di Firenze
Matteo Lapucci MATTEO.LAPUCCI@UNIFI.IT
Dipartimento di Ingegneria dell'Informazione
Università di Firenze
Chih-Jen Lin CJLIN@CSIE.NTU.EDU.TW
Department of Computer Science
National Taiwan University
Marco Sciandrone MARCO.SCIANDRONE@UNIFI.IT
Dipartimento di Ingegneria dell'Informazione
Università di Firenze

Editor: Zaid Harchaoui

Abstract

In this work we present a novel way to solve the sub-problems that originate when using decomposition algorithms to train Support Vector Machines (SVMs). State-of-the-art Sequential Minimization Optimization (SMO) solvers reduce the original problem to a sequence of sub-problems of two variables for which the solution is analytical. Although considering more than two variables at a time usually results in a lower number of iterations needed to train an SVM model, solving the sub-problem becomes much harder and the overall computational gains are limited, if any. We propose to apply the two-variables decomposition method to solve the sub-problems themselves and experimentally show that it is a viable and efficient way to deal with sub-problems of up to 50 variables. As a second contribution we explore different ways to select the working set and its size, combining first-order and second-order working set selection rules together with a strategy for exploiting cached elements of the Hessian matrix. An extensive numerical comparison shows that the method performs considerably better than state-of-the-art software.

Keywords: SVM, Support Vector Machines, Decomposition Method, Working Set Selection, Sequential Minimal Optimization .

1. Introduction

Support Vector Machines (Boser et al., 1992; Cortes and Vapnik, 1995) have been widely and effectively employed for binary classification for the last two decades, often achieving state of arts results in different contexts. Given a data set of n training vectors $v_i \in \mathbb{R}^m$ and their associated labels $y_i \in \{-1, 1\}$, training an SVM amounts to solve a convex optimization problem. Its dual formulation, which is often preferred for easily employing non-linear kernels, consists of the following linearly and bound constrained quadratic optimization

©2021 Giulio Galvan, Matteo Lapucci, Chih-Jen Lin and Marco Sciandrone.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/. Attribution requirements are provided at http://jmlr.org/papers/v22/19-632.html.

problem:

$$\begin{aligned} \min_{\alpha} \; & f(\alpha) = \frac{1}{2}\alpha^T Q \alpha - \alpha^T e \\ \text{s.t.} \; & y^T \alpha = 0, \\ & 0 \leq \alpha \leq C, \end{aligned} \tag{1}$$

where e is the n-dimensional vector of all ones, $\alpha \in \mathbb{R}^n$ and $Q \in \mathbb{R}^{n \times n}$ is positive semi-definite. Matrix Q is given by $Q_{ij} = y_i y_j K_{ij}$, where K is the kernel matrix with $K_{ij} = \phi(v_i)^T \phi(v_j)$, for some mapping ϕ to a higher (possibly infinite) dimensional space.

Problem (1) has been extensively studied (see the work of Bottou et al., 2007 or Piccialli and Sciandrone, 2018 for an extensive survey on optimization methods for SVM training) by both the machine learning and optimization communities over the years especially as data sets are becoming ever larger. Indeed, when the number of data point n is huge (as in many big data applications) the Hessian matrix Q, which is dense, cannot be fully stored in memory so that standard methods for quadratic programming cannot be used.

Decomposition methods, which are the focus of this work, are particularly well suited to deal with this issue, since the original problem is divided into a sequence of smaller subproblems obtained by fixing subsets of variables. The key aspects of a decomposition algorithm are hence 1) how we decompose the problem in smaller sub-problems and 2) how we solve the subproblems themselves.

The contribution of this work is thus twofold. Firstly we study how to effectively solve the subproblems. In particular we focus on solving subproblems of more than two variables, which originate from the novel working set selection rule that is the second contribution of this work.

The paper is organized as follows. We review the literature on decomposition methods for SVM in Section 2. Then we summarize the proposed algorithm in Section 3 before giving the details on solution of the subproblem in Section 3.1 and on the novel working set selection rule in Section 3.2. Extensive numerical experiments are presented in Section 4, where, after providing setup details (Section 4.1) and giving a brief description of performance profiles (Section 4.2) we analyze first the efficiency of the solver for the subproblem (Section 4.3), then the performance of different working set selection rules (Sections 4.4, 4.5 and 4.6), before comparing the whole algorithm against the state-of-art solver LIBSVM (Chang and Lin, 2011) in Section 4.7. We also provide a brief additional analysis about the relevance of the cache size in Section 4.8. Finally, we give some concluding remarks in Section 5.

2. Decomposition methods for SVM training

The literature on decomposition methods is wide and their convergence properties very well understood. General decomposition algorithms, however, are either applied to unconstrained optimization problems or when the feasible set has a simple structure, for instance the Cartesian product of subsets in smaller spaces. In SVM training problems, however, the feasible set cannot be simply partitioned into blocks and hence, custom decomposition methods are needed.

In a decomposition method for SVM training problems, at each iteration k two sets of indexes $W \subset \{1, \ldots, n\}$ (referred to as working set (WS)) and $\overline{W} = \{1, \ldots, n\} \setminus W$ are

identified, with $q = |W| \ll n$, and a subproblem of the following form is solved.

$$\begin{aligned} \min_{\alpha_W} \; & f(\alpha_W, \alpha^k_{\overline{W}}) = \frac{1}{2}\alpha_W^T Q_{WW}\alpha_W + p_W^T \alpha_W \\ \text{s.t. } & y_W^T \alpha_W = -y_{\overline{W}}^T \alpha^k_{\overline{W}}, \\ & 0 \le \alpha_W \le C, \end{aligned} \tag{2}$$

where $p_W = -e_W + Q_{W\overline{W}}\alpha_{\overline{W}}$. The feasible set $\mathcal{F}(\alpha_{\overline{W}})$ for the sub-problem is given by $\{\alpha_W \in \mathbb{R}^q \mid 0 \le \alpha_W \le C,\ y_W^T\alpha_W = -y_{\overline{W}}^T\alpha_{\overline{W}}\}$. To construct the subproblem, exploiting the symmetry of Q, columns Q_W of the Hessian matrix corresponding to the indexes in W are needed:

$$Q_W = \begin{bmatrix} Q_{WW} \\ Q_{\overline{W}W} \end{bmatrix} \tag{3}$$

The general framework of a decomposition scheme is described in Algorithm 1.

Algorithm 1 General Decomposition Framework for SVM Training Problems

Input: $\alpha^0 = 0$, $\nabla f(\alpha^0) = -e$, $k = 0$

1: **while** *the stopping criterion is not satisfied* **do**
2: select the working set W^k
3: retrieve the columns Q_W identified by (3)
4: set $W = W^k$ and compute a solution $\alpha^\star_W$ of subproblem (2)
5: set $\alpha_i^{k+1} = \begin{cases} \alpha_i^\star & \text{for } i \in W \\ \alpha_i^k & \text{otherwise} \end{cases}$
6: set

$$\nabla f(\alpha^{k+1}) = \nabla f(\alpha^k) + Q_W(\alpha^{k+1} - \alpha^k) = \nabla f(\alpha^k) + \sum_{i \in W} Q_i(\alpha_i^{k+1} - \alpha_i^k)$$

where Q_i is the i-th column of Q
7: set $k = k + 1$
8: **end while**
9: **return** $\alpha^\star = \alpha^k$

In terms of computational cost, the most expensive step at each iteration of a decomposition method is, especially for large data sets, the computation of the columns of the Hessian matrix corresponding to the indices in the working set W. In particular, if each K_{ij} costs $\mathcal{O}(m)$ (as with typically employed kernels), the cost for one column is $\mathcal{O}(nm)$. These columns are needed in (3) for setting up subproblems (2) and then at step 6 of Algorithm 1 for updating the gradients. The other operations that may require a computational effort are, as we will clarify in the following, the working set selection procedure itself, the solution of subproblems and the update of gradients.

The cardinality q of the working set, i.e. the dimension of the sub-problem, has to be strictly greater than 1, otherwise we would have $\alpha^{k+1} = \alpha^k$. Based on q, two cases can be distinguished:

- Sequential Minimal Optimization (SMO) algorithms, where $q = 2$;

- General Decomposition Algorithms, where $q > 2$.

The main difference between SMO and General Decomposition Algorithms lies in how the sub-problems are solved. When $q = 2$, variables can be updated by an analytical formula, as shown by Platt (1999), whereas for larger working sets the adoption of an iterative method is necessary, e.g., the software MINOS (Osuna et al., 1997), the LOQO primal-dual interior point method (Saunders et al., 1998; Vanderbei, 1999; Joachims, 1999) or gradient projection (Dai and Fletcher, 2006; Zanni, 2006). In General Decomposition Algorithms, the working set rarely contains more than a few tens of variables, otherwise a single iteration might become too expensive. In fact, the inefficiency and complexity of QP solvers is the main reason why researchers stopped using working sets of $q > 2$ variables and the interest in SMO algorithms arose.

Concerning the working set selection rule (WSSR), let us first consider SMO algorithms, whose related literature is older and wider. At a feasible point α, the index sets

$$\begin{aligned} R(\alpha) &= \{h \in \{1,\dots,n\} \mid 0 < \alpha_h < C \ \vee \ (\alpha_h = 0 \wedge y_h = 1) \ \vee \ (\alpha_h = C \wedge y_h = -1)\} \\ S(\alpha) &= \{h \in \{1,\dots,n\} \mid 0 < \alpha_h < C \ \vee \ (\alpha_h = 0 \wedge y_h = -1) \ \vee \ (\alpha_h = C \wedge y_h = 1)\} \end{aligned} \tag{4}$$

characterize the status of the variables. In particular, it can be shown (see, e.g., Lin, 2002b) that α is a solution of (1) if and only if

$$\max_{i \in R(\alpha)} \{-y_i \nabla f(\alpha)_i\} \leq \min_{j \in S(\alpha)} \{-y_j \nabla f(\alpha)_j\}. \tag{5}$$

Given a non-optimal, feasible point α, a pair $\{i, j\} \in R(\alpha) \times S(\alpha)$ such that

$$-y_i \nabla f(\alpha)_i > -y_j \nabla f(\alpha)_j$$

is referred to as a violating pair. If

$$i^\star \in \arg\max_{i \in R(\alpha)} \{-y_i \nabla f(\alpha)_i\} \qquad j^\star \in \arg\min_{j \in S(\alpha)} \{-y_j \nabla f(\alpha)_j\}, \tag{6}$$

then $\{i^\star, j^\star\}$ is called a most violating pair (MVP).

A classical way of choosing the working set for SMO is choosing a most violating pair (Joachims, 1999; Keerthi et al., 2001). This selection is cheap to compute, as it is done in $\mathcal{O}(n)$ time, and it can be proved that SMO algorithm with this working set selection strategy is globally convergent (Lin, 2001, 2002a). In the following we will refer to this rule as WSS1. WSS1 is based on first order information: indeed, $i^\star$ and $j^\star$ identify the direction with only two non-null (unit) components that minimizes the first order approximation

$$f(\alpha^k + d) \simeq f(\alpha^k) + \nabla f(\alpha^k)^T d.$$

Other working set selection rules with global convergence properties are described by Chen et al. (2006), Chang et al. (2000), Lin et al. (2009), Lucidi et al. (2007) and Fan et al. (2005). Amongst those, the most widely employed one in practice (e.g., in LIBSVM) is the one from Fan et al. (2005), which exploits second order information. We will refer to this rule as WSS2. Since f is quadratic, the exact reduction of the objective value is given by:

$$f(\alpha^k) - f(\alpha^k + d) = -\nabla f(\alpha^k)^T d - \frac{1}{2} d^T \nabla^2 f(\alpha^k) d. \tag{7}$$

Unfortunately, searching the pair of indices that identify the feasible direction with two non-zero components maximizing the objective decrease requires $\mathcal{O}(n^2)$ operations and is thus impractical. The idea is therefore that of choosing one variable as in WSS1 and, having that fixed, identifying the other index so that (7) is maximized. This is done, assuming the kernel is positive-definite, by setting

$$\begin{aligned} i &\in \arg\max_{t \in R(\alpha^k)} \{-y_t \nabla f(\alpha^k)_t\} \\ j &\in \arg\min_{h \in S(\alpha^k)} \left\{ -\frac{\delta_{ih}^2}{\rho_{ih}} \mid -y_h \nabla f(\alpha^k)_h < -y_i \nabla f(\alpha^k)_i \right\}, \end{aligned} \tag{8}$$

where

$$\rho_{ih} = K_{ii} + K_{hh} - 2K_{ih} \qquad \delta_{ih} = -y_i \nabla f(\alpha^k)_i + y_h \nabla f(\alpha^k)_h, \tag{9}$$

being K the kernel matrix. This procedure has cost $\mathcal{O}(n)$, even though it is in fact slightly more expensive than WSS1.

The notion of MVP gives also rise to the simple and widely employed stopping criterion

$$m(\alpha^k) \leq M(\alpha^k) + \epsilon, \tag{10}$$

where $\epsilon > 0$ and

$$m(\alpha) = \max_{h \in R(\alpha)} -y_h \nabla f(\alpha)_h \qquad M(\alpha) = \min_{h \in S(\alpha)} -y_h \nabla f(\alpha)_h. \tag{11}$$

Note that $m(\alpha) \leq M(\alpha)$ is exactly the optimality condition (5). It has been proved (Lin, 2002b) that all rules selecting *constant-factor* violating pairs (Chen et al., 2006) generate sequences $\{\alpha^k\}$ such that $m(\alpha^k) - M(\alpha^k) \to 0$, i.e. algorithms of this type satisfy stopping criterion (10) in a finite number of iterations for any $\epsilon > 0$. A violating pair $\{i, j\}$ is referred to as a constant-factor violating pair if

$$y_i \nabla f(\alpha^k)_i - y_j \nabla f(\alpha^k)_j \leq \sigma \left(y_{i^\star} \nabla f(\alpha^k)_{i^\star} - y_{j^\star} \nabla f(\alpha^k)_{j^\star} \right), \tag{12}$$

being $0 < \sigma \leq 1$ and $\{i^\star, j^\star\}$ the MVP. WSS1 and WSS2 are indeed instances of the constant-factor violating pair rule and thus lead to finite termination.

In Algorithm 2 we summarize the SMO decomposition method with WSS1, also referred to as SMO-MVP algorithm.

Algorithm 2 SMO-MVP Algorithm

Input: $\alpha^0 = 0$, $\nabla f(\alpha^0) = -e$, $k = 0$

while $m(\alpha^k) - M(\alpha^k) > \epsilon$ **do**

  select the working set $W^k = \{i^k, j^k\}$ according to

$$i^k \in \arg\max_{i \in R(\alpha^k)} -y_i \nabla f(\alpha^k)_i$$

$$j^k \in \arg\min_{j \in S(\alpha^k)} -y_j \nabla f(\alpha^k)_j$$

  set $W = W^k, \bar{W} = \{1, \dots, n\} \setminus W$ and analytically compute a solution $\alpha^\star_W$ of sub-problem (2)

  set $\alpha_h^{k+1} = \begin{cases} \alpha_h^\star & \text{for } h \in W \\ \alpha_h^k & \text{otherwise} \end{cases}$

  set $\nabla f(\alpha^{k+1}) = \nabla f(\alpha^k) + Q_{i^k}(\alpha_{i^k}^{k+1} - \alpha_{i^k}^k) + Q_{j^k}(\alpha_{j^k}^{k+1} - \alpha_{j^k}^k)$

  set $k = k + 1$

end while

return α^k

As for General Decomposition Methods, a first working set selection rule has been proposed by Joachims (1999) and the asymptotic convergence of the decomposition method based on such rule is proved by Lin (2001). The finite termination is shown by Lin (2002b). The scheme used in the SVM^{light} software, selects a working set W with an even number q of variables by solving the following problem:

$$\begin{aligned} \min_d \ & \nabla f(\alpha^k)^T d \\ \text{s.t. } & y^T d = 0, \quad -1 \le d_i \le 1 \ \ \forall\, i = 1, \dots, n, \\ & d_i \ge 0 \text{ if } \alpha_i^k = 0, \quad d_i \le 0 \text{ if } \alpha_i^k = C, \\ & |\{d_i \mid d_i \ne 0\}| \le q. \end{aligned} \tag{13}$$

With the above problem, the steepest feasible direction with at most q non-zero components is identified. Optimal W according to (13) can be efficiently found, similarly as for WSS1, by selecting the $q/2$ most violating pairs. In fact, WSS1 is the particular case of (13) when $q = 2$. This selection strategy guarantees global convergence, at the cost of a limited freedom in choosing variables. We will also refer to this WSSR in the following as extended-WSS1.

The theoretical issue about convergence with more arbitrary selection rules is open. Asymptotic convergence to optimal solutions has not been proven in this scenario. However, Zhang et al. (2018) proved finite termination of Algorithm 1 with stopping criterion (10) under the assumption that at each iteration at least one pair of indexes $\{i, j\} \in R(\alpha^k) \times S(\alpha^k)$ is present in the working set such that

$$-y_i \nabla f(\alpha^k)_i > -y_j \nabla f(\alpha^k)_j + \epsilon.$$

This result improves the work of Takahashi and Nishi (2006), where relaxed definitions of sets $R(\alpha^k)$ and $S(\alpha^k)$ are considered.

Note that, by the above result, any working set selection rule that inserts the MVP into the WS is guaranteed to generate a finite sequence of iterates. This fact is quite important, allowing to consider sophisticated mixed selection strategies without any risk of non-termination. In particular, selection rules can safely be considered that better exploit the commonly used caching technique, consisting of storing the recently used columns of the Hessian matrix in order to avoid their recalculation. Glasmachers and Igel (2006), Lucidi et al. (2009) and Lin et al. (2009) have studied decomposition methods designed to couple convergence properties and the exploitation of the caching strategy. Serafini and Zanni (2005) proposed an experimentally efficient way of choosing the working set based on MVPs and caching, but asymptotic convergence properties have not been proven for this rule.

2.1 Related methods for SVM training

An SVM dual problem slightly different from (1) has been suggested by Frie et al. (1998), where the parameter C is added to the diagonal of Q and α_i becomes unbounded, i.e.,

$$\begin{aligned} \min_{\alpha} \ & \frac{1}{2}\alpha^T \left(Q + \frac{I}{C}\right) \alpha - e^T \alpha \\ \text{s.t. } & \alpha \geq 0, \\ & y^T \alpha = 0, \end{aligned} \tag{14}$$

where I is the identity matrix. This problem is very close to (1), so most optimization methods developed for one can be applicable to the other. An example is the decomposition method studied in this work. On the other hand, some methods specific for (14) have been proposed. For example, (14) can be converted into the problem of computing the nearest points between two convex polytopes and solved by certain iterative algorithms (Keerthi et al., 2000).

The general decomposition framework in Algorithm 1 considers a fixed (or bounded) size of the working set throughout iterations. In another line of research, some works (Vishwanathan and Murty, 2002; Abe, 2008) take the property that support vectors are a subset of data points and dynamically adjust the working set to eventually cover all support vectors (i.e., all data points with $\alpha_i > 0$ at an optimal α). A concern of such approaches is that for large problems with many support vectors, the complexity of solving each sub-problem may be prohibitive.

A more significant change of the SVM dual problem, leading to a formulation only having box constraints, was proposed by Mangasarian and Musicant (1999). A Successive Over-relaxation scheme was proposed to solve the problem in the aforementioned work, while Hsu and Lin (2002) designed a specific decomposition method, making use of a specialized version of TRON (Lin and Moré, 1999) to solve the subproblems.

3. The proposed algorithm

In this work we propose a new variant of the non-SMO decomposition method for kernel SVMs training which is based on two main ideas:

Algorithm 3 Two-Level Decomposition Method for SVM Training

Input: $\alpha^0 = 0$, $\nabla f(\alpha^0) = -e$, $k = 0$, $q \geq 4$

while $m(\alpha^k) - M(\alpha^k) > \epsilon$ **do**
  select 4 variables to define the working set $W \subset \{1, \ldots, n\}$ according to (21)
  eventually add $q-4$ variables to W, corresponding to cached columns of the Hessian matrix by Algorithm 5.
  compute α_W^{k+1} by applying Algorithm 4 to $\min_{\alpha_W \in \mathcal{F}(\alpha_{\overline{W}}^k)} f(\alpha_W, \alpha_{\overline{W}}^k)$
  $\alpha_{\overline{W}}^{k+1} = \alpha_{\overline{W}}^k$
  $\nabla f(\alpha^{k+1}) = \nabla f(\alpha^k) + \sum_{h \in W} Q_h(\alpha_h^{k+1} - \alpha_h^k)$
  set $k = k + 1$
end while
return α^k

- the q-variables sub-problems are solved by means of an inner SMO procedure, so that no line-search procedures are required;
- a novel working set selection rule for $q > 2$ variables that exploits both first and second order information is employed.

An outline of the algorithm is summarized by the pseudocode in Algorithm 3.

We first detail the solution of the subproblems and then describe the novel proposed working set selection rule.

3.1 Solving the subproblems

The subproblem with the q variables in the working set W generated at each iteration k by the decomposition procedure is given by (2). For the sake of simplicity, let us change the notation to

$$\begin{aligned} \min_x \ & \frac{1}{2} x^T \tilde{Q} x + p^T x \\ \text{s.t. } & a^T x = b, \\ & 0 \leq x \leq C. \end{aligned} \tag{15}$$

Except for the fact that the constant term of the equality constraint is not zero and $p \neq e$, the subproblems have exactly the same form as (1). Thus, classical SVM decomposition schemes can be applied also to the subproblems. In fact, we will show that SMO technique works extremely well not only on large problems, but also with small problems such as the subproblems here.

The efficiency of SMO algorithms mainly comes from the fact that all the updates of the variables are performed in closed form, not requiring the computational effort of a line-search algorithm.

These observations led us to adopt SMO-MVP to solve the subproblems. We can hence see the whole training algorithm as a two-level decomposition scheme, where the inner subproblems are solved by Algorithm 2. In the following, we will also refer to this two-level decomposition scheme as TLD-ISMO (Two Level Decomposition with Inner Sequential Minimal Optimization).

It should be noted that the idea of a double decomposition have been considered before in the literature, although in different contexts or with different intents. In the work of Pérez-Cruz et al. (2004) an SVM instance is solved by repeatedly forming a sub-problem consisting in a "large chunk" of thousands of examples. This large sub-problem is solved with a decomposition algorithm itself (this time exploiting only a few variables at a time). At each iteration the large chunk is updated with the samples which do not satisfy optimality conditions until convergence.

Perhaps more closely related to our work is the analysis to be found in Yu et al. (2011). In this study the focus is on logistic regression and maximum entropy models. The authors propose to solve the problems in the dual space with decomposition methods as is done for SVMs problems. The resulting sub-problems, however, are significantly different from the sub-problems that originates in the case of SVM. In particular the authors note that is possible to solve the sub-problems with a coordinate descent decomposition algorithm (which cannot be used in SVM since at least 2 variables are to be moved at a time).

Our subproblems solver is described in detail in Algorithm 4. Let $W = \{h_1, \dots, h_q\}$ so that x_i corresponds to α_{h_i} and a_i corresponds to y_{h_i}.

Given a feasible point x for the subproblem, we can define, similarly as (4), the sets

$$\begin{aligned} \tilde{R}(x) &= \{\ell \in \{1, \dots, q\} \mid 0 < x_\ell < C \ \vee \ (x_\ell = 0 \wedge a_\ell = 1) \vee (x_\ell = C \wedge a_\ell = -1)\}, \\ \tilde{S}(x) &= \{\ell \in \{1, \dots, q\} \mid 0 < x_\ell < C \ \vee \ (x_\ell = 0 \wedge a_\ell = -1) \vee (x_\ell = C \wedge a_\ell = 1)\}. \end{aligned} \tag{16}$$

Then, the most violating pair $\{i, j\} \in \tilde{R}(x) \times \tilde{S}(x)$ at x can be defined, similarly as (6), as

$$i = \arg\max_{\ell \in \tilde{R}(x)} \{-a_\ell((\tilde{Q}x)_\ell + p_\ell)\} \qquad j = \arg\min_{\ell \in \tilde{S}(x)} \{-a_\ell((\tilde{Q}x)_\ell + p_\ell)\}. \tag{17}$$

Finally, we can introduce the quantities

$$\tilde{M}(x) = \max_{\ell \in \tilde{R}(x)} \{-a_\ell((\tilde{Q}x)_\ell + p_\ell)\} \qquad \tilde{m}(x) = \min_{\ell \in \tilde{S}(x)} \{-a_\ell((\tilde{Q}x)_\ell + p_\ell)\}, \tag{18}$$

similarly as (11), in order to define the stopping criterion

$$\tilde{m}(x) \leq \tilde{M}(x) + \epsilon_{\text{in}}, \tag{19}$$

where ϵ_{in} is a tolerance.

At each inner iteration κ we select two variables from W via WSS1. The small dimension of the subproblems may not require more sophisticated selection rules. We then solve the sub-subproblem

$$\begin{aligned} &\min_{x_i, x_j} \frac{1}{2} \begin{pmatrix} x_i & x_j \end{pmatrix} \begin{pmatrix} \tilde{Q}_{ii} & \tilde{Q}_{ij} \\ \tilde{Q}_{ji} & \tilde{Q}_{jj} \end{pmatrix} \begin{pmatrix} x_i \\ x_j \end{pmatrix} + \sum_{\substack{\ell=1 \\ \ell \neq i,j}}^{q} (\tilde{Q}_{\ell j} x_\ell^\kappa x_j + \tilde{Q}_{\ell i} x_\ell^\kappa x_i) + p_i x_i + p_j x_j \\ &\text{s.t. } a_i x_i + a_j x_j = a_i x_i^\kappa + a_j x_j^\kappa \\ &\qquad 0 \leq x_i, x_j \leq C, \end{aligned} \tag{20}$$

which can be done in closed form as described, e.g., by Chang and Lin (2011). The sub-procedure employed to solve the subproblems is summarized in Algorithm 4. It is worth

remarking that, differently from the case of problem (1), matrix $\tilde{Q}$ is fully available to Algorithm 4 that solves (15).

Algorithm 4 Inner SMO Algorithm

Input: $x^0 = \alpha_W^k$, $\kappa = 0$
1: **while** $\tilde{m}(x^\kappa) - \tilde{M}(x^\kappa) > \epsilon_{\text{in}}$ **do**
2: select $\{i, j\} \subset W$ according to (17)
3: compute $x_i^{\kappa+1}$, $x_j^{\kappa+1}$ by solving analytically problem (20)
4: set $x_\ell^{\kappa+1} = x_\ell^\kappa$ for all $\ell \neq i, j$
5: update the gradients
6: set $\kappa = \kappa + 1$
7: **end while**
8: **return** $x^{\kappa+1}$

The inner solver stops when KKT conditions are (approximately) satisfied for the subproblem. Note that, since we employ Algorithm 2, this is proven to happen in a finite number of iterations for any $\epsilon_{\text{in}} > 0$. Also note that, although x^0 may be initialized with any feasible point, it is particularly useful in practice starting from the current estimate.

3.2 A novel working set selection rule

SMO algorithms are widely considered the state-of-the-art decomposition methods for Support Vector Machines training (see e.g. Piccialli and Sciandrone (2018)), and WSS2 introduced by Fan et al. (2005) is accepted as more efficient than WSS1 originally introduced by Keerthi et al. (2001) although numerical experience shows that, while WSS2 is indeed generally better in terms of number of iterations, WSS1 can still be competitive in terms of computational time in some cases (as numerical evidence will show in Section 4.5).

The core idea of the proposed working set selection rule is, thus, leveraging on the efficiency of the subproblem solver, to exploit the benefits of both WSS1 and WSS2 selecting two variables according to WSS1 and other two variables according to WSS2. Formally, we choose the working set $W = \{i_1, i_2, j_1, j_2\}$ as follows:

$$\begin{aligned} i_1 &= \arg\max_{h \in R(\alpha^k)} \left\{ -y_h \nabla_h f(\alpha^k) \right\}, \\ j_1 &= \arg\min_{h \in S(\alpha^k)} \left\{ -y_h \nabla_h f(\alpha^k) \right\}, \\ i_2 &= \arg\max_{h \in R(\alpha^k), h \neq i_1} \left\{ -y_h \nabla_h f(\alpha^k) \right\}, \\ j_2 &= \arg\min_{h \in S(\alpha^k), h \neq j_1} \left\{ -\frac{\delta_{i_2 h}^2}{p_{i_2 h}} \;\middle|\; -y_h \nabla_h f(\alpha^k) < -y_{i_2} \nabla_{i_2} f(\alpha^k) \right\}. \end{aligned} \tag{21}$$

We will refer to the working set selection rule (21) as WSS-MIX. As a matter of fact, swapping the order of the two pairs, i.e., selecting the first pair by WSS2 and the second one by WSS1, is a possibility. In fact, from preliminary experiments we observed that such change does not significantly affect the performance.

Algorithm 5 Working Set Filling

Input: W^k, $|W^k| = \bar{q}$, W^{k-1}, $|W^{k-1}| = q$

Let

$$F = \{h \in W^{k-1} \mid 0 < \alpha_h^k < C\}$$
$$L = \{h \in W^{k-1} \mid \alpha_h^k = 0\}$$
$$U = \{h \in W^{k-1} \mid \alpha_h^k = C\}$$

while $|W^k| \leq q$ **and** $F \setminus W^k \neq \emptyset$ **do**
  Add to W^k the index $h \in F \setminus W^k$ that has been in the working set for the least number of iterations.
end while
while $|W^k| \leq q$ **and** $L \setminus W^k \neq \emptyset$ **do**
  Add to W^k the index $h \in L \setminus W^k$ that has been in the working set for the least number of iterations.
end while
while $|W^k| \leq q$ **and** $U \setminus W^k \neq \emptyset$ **do**
  Add to W^k the index $h \in U \setminus W^k$ that has been in the working set for the least number of iterations.
end while

A crucial technique employed in all of state-of-art decomposition method for SVM is to cache the most recently computed Hessian matrix columns for later reuse, thus saving up in the number of kernel computations. Moreover it is has been shown to be beneficial, especially for large data sets, to make use of the cached columns by augmenting the working set with additional variables of which the correspondent Hessian columns are currently cached (Serafini and Zanni, 2005). This idea fits nicely in our framework, since we are able to deal with large working sets of variables. We thus enhance WSS-MIX by adding cached variables.

A refined version of the rule from Serafini and Zanni (2005) is proposed by Zanni (2006). By such a rule, the working set is filled as follows: $\bar{q}$ variables are selected according to extended-WSS1. Then they add to the working set $q - \bar{q}$ variables as described by Algorithm 5. Priority is given to free variables, then lower bound variables and last upper bound variables. The idea is that upper bound variables have probably reached their final value, while free variables are completing the optimization process. Also, indexes that have been in the working set for less iterations are preferred. For our method, we borrow Algorithm 5 to complete the working set after the first 4 variables are selected according to WSS-MIX.

Zanni (2006) fixes arbitrarily the size of the working set at the beginning and then adaptively tunes the proportion of variables that are selected with extended-WSS1 and with Algorithm 5. Here we follow a slightly different approach and keep both the number of variables chosen with WSS-MIX and of cached variables fixed from the beginning.

Identifying the ideal number of extra variables to add at each iteration for each problem is not an easy task. It seems unlikely that choosing more than 20 variables is useful; indeed, if the data set is very large, the cache will typically not be large enough to store more

than $\sim$ 20 - 30 variables, while if the data set is not enough large, the time saved by not recomputing kernels is lost in performing unnecessary computation. Furthermore, the higher cost of solving subproblems is another reason for not using large values of q. Indeed, in past non-SMO works q have usually been set around 10 or 20 (Joachims, 1999; Hsu and Lin, 2002).

We found that a rule of thumb for choosing the number of additional variables with a problem from a given data set can be obtained starting from the following formula. Let

$$S = \frac{\text{cachesize}}{8 \times n^2 \times m} \tag{22}$$

be the fraction of Hessian matrix elements that can be cached (assuming elements are double precision values) divided by the number of features (the cost of computing one element of the Hessian is proportional to this quantity). The rule is as follows:

- if $S > 10^{-3}$ do not select additional variables,
- if $10^{-3} < S < 10^{-5}$ select 6 additional variables to the working set by means of Algorithm 5,
- if $S < 10^{-5}$ select 14 additional variables to the working set by means of Algorithm 5.

The idea is that the addition of cached variables is useful if variables often get out of cache before being used again and if kernel computation is expensive; note the cost of computing one Hessian column is $\mathcal{O}(mn)$, where n is the number of examples in the data set and m the number of features. Clearly, situations where the cache size is too small to store the number of additional variables resulting from the above rule may happen; in such cases the number of selected variables should be equal to the cache capacity.

3.3 Convergence properties of the proposed algorithm

To start the discussion we note that the algorithm is well defined, i.e. the inner loop (Algorithm 4) stops in a finite number of steps. This trivially comes from the finite termination property of SMO-MVP algorithm (Lin, 2002a,b).

Finite termination of the whole algorithm is, instead, still an open issue. Indeed WSS-MIX (both with or without cached variables) does not satisfy the sufficient conditions for asymptotic global convergence and finite termination stated by Lin (2002a,b). Under the assumption that the sub-problem are solved exactly, however, finite termination of the procedure is guaranteed by the result proven by Zhang et al. (2018). Nonetheless, the proposed algorithm solves the sub-problems up to ϵ_{in} precision and, since $M(\alpha)$ and $m(\alpha)$ are not continuous, even letting $\epsilon_{\text{in}} \to 0$ does not guarantee that the result from Zhang et al. (2018) holds. We can however show that

1. the objective function is monotonically non increasing;
2. at each iteration the variables are updated;

3. the working set changes at every iteration, so that it cannot happen that the algorithm infinitely loops on the same sub-problem.

Point 1. is trivial, we prove points 2. and 3.

Let W_k be the working set at the (outer) iteration k. Let ϵ_{in} and ϵ be the stopping tolerances of the inner and outer loops respectively, with $\epsilon_{\text{in}} \leq \epsilon$.

Let us define

$$\bar{R}(\alpha^k_{W_k}) = W_k \cap R(\alpha^k) \qquad \bar{S}(\alpha^k_{W_k}) = W_k \cap S(\alpha^k).$$

From the above definition, we have that if $\{i,j\} \subset W_k$ and $\{i,j\} \in R(\alpha^k) \times S(\alpha^k)$, then $\{i,j\} \in \bar{R}(\alpha^k_{W_k}) \times \bar{S}(\alpha^k_{W_k})$.

From the instructions of the algorithm and the definition of WSS-MIX, we have that at the beginning of iteration k there exists $\{i,j\} \in \bar{R}(\alpha^k_{W_k}) \times \bar{S}(\alpha^k_{W_k})$ such that

$$-y_i \nabla f(\alpha^k)_i + y_j \nabla f(\alpha^k) > \epsilon. \tag{23}$$

Moreover, from the instructions of Algorithm 4, at the end of the k-th outer iteration we get α^{k+1} such that, for all $\{i,j\} \in \bar{R}(\alpha^{k+1}_{W_k}) \times \bar{S}(\alpha^{k+1}_{W_k})$, it holds

$$-y_i \nabla f(\alpha^{k+1})_i + y_j \nabla f(\alpha^{k+1}) \leq \epsilon_{\text{in}} \leq \epsilon. \tag{24}$$

If it was $\alpha^k = \alpha^{k+1}$, (23) and (24) would be in contradiction. Therefore we have that $\alpha^{k+1} \neq \alpha^k$, i.e., point 2. holds.

From the definition of WSS-MIX, we also know that the working set W_{k+1} contains a pair $\{i_{k+1}, j_{k+1}\} \in R(\alpha^{k+1}) \times S(\alpha^{k+1})$ such that

$$-y_{i_{k+1}} \nabla f(\alpha^{k+1})_{i_{k+1}} + y_{j_{k+1}} \nabla f(\alpha^{k+1})_{j_{k+1}} > \epsilon, \tag{25}$$

otherwise the outer stopping criterion would have been satisfied. We will prove that $\{i_{k+1}, j_{k+1}\} \nsubseteq W_k$, so $W_{k+1} \neq W_k$. Assume by contradiction that $\{i_{k+1}, j_{k+1}\} \subset W_k$. Then, since $\{i_{k+1}, j_{k+1}\} \in R(\alpha^{k+1}) \times S(\alpha^{k+1})$, it has to be $\{i_{k+1}, j_{k+1}\} \in \bar{R}(\alpha^{k+1}_{W_k}) \times \bar{S}(\alpha^{k+1}_{W_k})$. Thus, both (24) for $\{i,j\} = \{i_{k+1}, j_{k+1}\}$ and (25) hold, which is absurd. Point 3. is thus proved.

4. Numerical experiments

In this section we provide numerical evidence to back up each component of the proposed algorithm before comparing the efficiency of the method as a whole. We use several data sets of different dimensions from the LIBSVM collection to obtain different instances of SVM training problems for several values of the hyper-parameters. We detail the common setup of all the experiments in the next Section.

4.1 Benchmark problems and experiments setup

All the trials were carried out on the same computer (Intel Xeon CPU 12 cores, 16 GB RAM). All implementations of the decomposition methods for SVM training considered in

Data set	Training Size	Features	Class
`splice`	1,000	60	small
`a1a`	1,605	123	small
`leukemia`	38	7,129	small
`a9a`	32,561	123	medium size
`w8a`	49,749	300	medium-size
`ijcnn1`	49,990	22	medium-size
`rcv1.binary`	20,242	47,236	large
`real-sim`	72,309	20,958	large
`covtype.binary`	581,012	54	huge

Table 1: Details of the 9 initial data sets.

this work include the use of a cache of size 100MB to store Hessian matrix values; note that 100MB were always enough to store at least 20 Hessian columns in the problems considered in the presented experiments. Also, kernels computation procedure has always been parallelized on 10 cores, for accelerating the experimental process. The effects of the shrinking heuristic (Joachims, 1999; Fan et al., 2005; Chang and Lin, 2011) often applied by SVM solvers, were not investigated; we thus set it off in all the algorithms used in the experiments of this work.

We initially considered a collection of 9 data sets for binary classification, taken from LIBSVM data sets collection webpage [1]. These data sets were used to evaluate the performances during the development of the new algorithm. In selecting these data sets we aimed at building a test suite that could cover a variety of situations in terms of data set dimensions. In Table 1 we list them, specifying their characteristics, i.e., number of examples and features.

The data sets `a1a`, `a9a` and `covtype.binary` are from the UCI Machine Learning Repository (Dua and Karra Taniskidou, 2017); binarization of problem `covtype` was done by Collobert et al. (2002); `a1a` and `a9a` have been compiled by Platt (1999); `w8a` is also from Platt (1999); `splice` comes from are from the Delve archive (`http://www.cs.toronto.edu/~delve`); `rcv1` has been published by Lewis et al. (2004). `ijcnn1` is the first of IJCNN 2001 challenge problems (Prokhorov, 2001); `leukemia` dataset comes from Golub et al. (1999), while `real-sim` original data can be found at `https://people.cs.umass.edu/~mccallum/data.html`.

To fairly measure the performance of the considered methods, we mimic a true application context. To do so, we built up our experimental benchmark as follows: for each data set, we generated a set of 25 RBF kernel SVM training problems, obtained by varying the values of the hyperparameters C and γ on a 5×5 grid. The grid takes values from $\{10^k \cdot \bar{C} \mid k = -2, -1, 0, 1, 2\}$ and $\{10^k \cdot \bar{\gamma} \mid k = -2, -1, 0, 1, 2\}$, where $\bar{C}$ and $\bar{\gamma}$ are suitable values found through a preliminary validation procedure conducted with LIBSVM. We recall that the RBF kernel is defined as

$$K(v, u) = \exp\left(-\gamma \|v - u\|^2\right).$$

1. `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html`

Data set	Training Size	Features
phishing	11,055	68
cod-rna	59,535	8
skin_nonskin	245,057	3
epsilon-subset	50,000	2,000
news20.binary	19,996	1,355,191
cifar-resnetv2-emb	60,000	1,024
SUSY-subset	500,000	18
HIGGS-subset	500,000	28

Table 2: Details of the 8 additional data sets.

Note that the classification of a data set in terms of dimensions (i.e., the data set scale) partially depends on the cache size. The diversity of datasets we employ allows us to simulate contexts where either the whole kernel matrix can be stored or repeated and expensive kernel computations are needed. We roughly grouped the datasets into four classes: small (a1a, splice, leukemia), medium-size (a9a, w8a, ijcnn1), large (rcv1, real-sim) and huge (covtype).

At the end of the development process, we selected 8 additional data sets and thus created 200 additional problems to evaluate the efficiency of our procedure. This was done in order to ensure that the good overall performance assessment of our method was not the result of an excessive specialization carried out during the development but that, on the contrary, our algorithm performs well in general, on new problems that could emerge in applications.

The additional 8 data sets are cod-rna (Uzilov et al., 2006), phishing (UCI), news20.binary (Keerthi and DeCoste, 2005), skin_nonskin (UCI), cifar-resnetv2-emb and subsets of epsilon, preprocessed as by Yuan et al. (2012), SUSY and HIGGS, both by Baldi et al. (2014). The data sets are again all taken from LIBSVM data sets collection webpage, except for the cifar-resnetv2-emb data set, which was obtained by training a ResNetv2 (He et al., 2016) convolutional neural network with 29 layers on the CIFAR-10 dataset (Krizhevsky, 2009) and extracting the embeddings before the final averaging layer. The network has been trained by SGD with early stopping on a 20% validation set extracted form the training part. The embeddings have been extracted after the training procedure for the whole training and test sets. The dimensions of the data sets are reported in Table 2. For these data sets, the values of $\bar{C}$ and $\bar{\gamma}$ were set respectively to the default 1 and $1/m$, in order to simulate applications.

In the experiment regarding the inner subproblems solvers, we employed for all algorithms stopping condition (19) with tolerance $\epsilon_{\text{in}} = 10^{-5}$.

As for the various versions of Algorithm 1, the implementation exploits the structure of LIBSVM (Chang and Lin, 2011). The considered methods share stopping criterion (10), with $\epsilon = 10^{-3}$, which is the default tolerance in LIBSVM. Different values of ϵ have also preliminarily been tried, leading to similar outcomes as $\epsilon = 10^{-3}$, i.e. the results of our computational study seem to be consistent w.r.t. different values of ϵ. The inner stopping tolerance for TLD-ISMO was again set to $\epsilon_{\text{in}} = 10^{-5}$, which satisfies the condition $\epsilon_{\text{in}} \leq \epsilon$ required in the analysis in Section 3.3. Such a value was selected, after a preliminary

experimental work, as a compromise. In fact, if the tolerance value is too small, a large amount of time is wasted at solving subproblems to unnecessary precision. On the other hand, too large values of ϵ_{in} lead to not exploiting enough the variables in the working set.

4.2 Performance profiles

In this work, the comparisons have mainly been carried out by means of performance profiles. Performance profiles have been proposed by Dolan and Moré (2002) as a mean to compare the performance of different solvers on a suite of test problems. Performance profiles provide a unified view of relative performance of the solvers that overcome the limitations of previous approaches: they do not involve interpreting long tables of values, nor do they suffer from the shortcomings of comparing, for instance, the number of wins, the average performance, quantiles, or other cumulative statistics (as argued by Dolan and Moré (2002)). For these reasons they have been adopted by an ever growing number of researches over the last decade.

Formally, consider a test suit of $\mathcal{P}$ problems and a set of solvers $\mathcal{S}$. For each solver $s \in \mathcal{S}$ and problem $p \in \mathcal{P}$ define

$$t_{p,s} = \text{the cost for solver } s \text{ to solve problem } p,$$

where cost is the performance metric we are interested in. For example the cost can be the number of function evaluations, the number of iterations or the CPU time. Define also the ratio

$$r_{p,s} = \frac{t_{p,s}}{\min_{s \in \mathcal{S}}\{t_{p,s}\}},$$

which expresses a relative measure of the performance on problem p of solver s against the performance of the best solver for this problem. If (and only if) a solver fails to solve a problem we put $r_{p,s} = r_M$ with $r_M \geq r_{p,s}\ \forall s, p$.

The performance profile for a solver s is thus the function

$$\rho_s(\tau) = \frac{1}{|\mathcal{P}|} \cdot |\{p \in \mathcal{P} \mid r_{p,s} \leq \tau\}|,$$

which represents the probability for solver s that a performance ratio $r_{p,s}$ is within a factor $\tau \in R$ of the best possible ratio. The function $\rho_s(\tau) : [1, +\infty] \to [0, 1]$ is, in fact, the cumulative distribution function for the performance ratio.

The value of $\rho_s(1)$ is the probability that the solver will win over the rest of the solvers while $\lim_{\tau \to r_M^-} \rho_s(\tau)$ is the probability that the solver solves a problem.

We are now ready to start our analysis, by focusing on the solution of the subproblems.

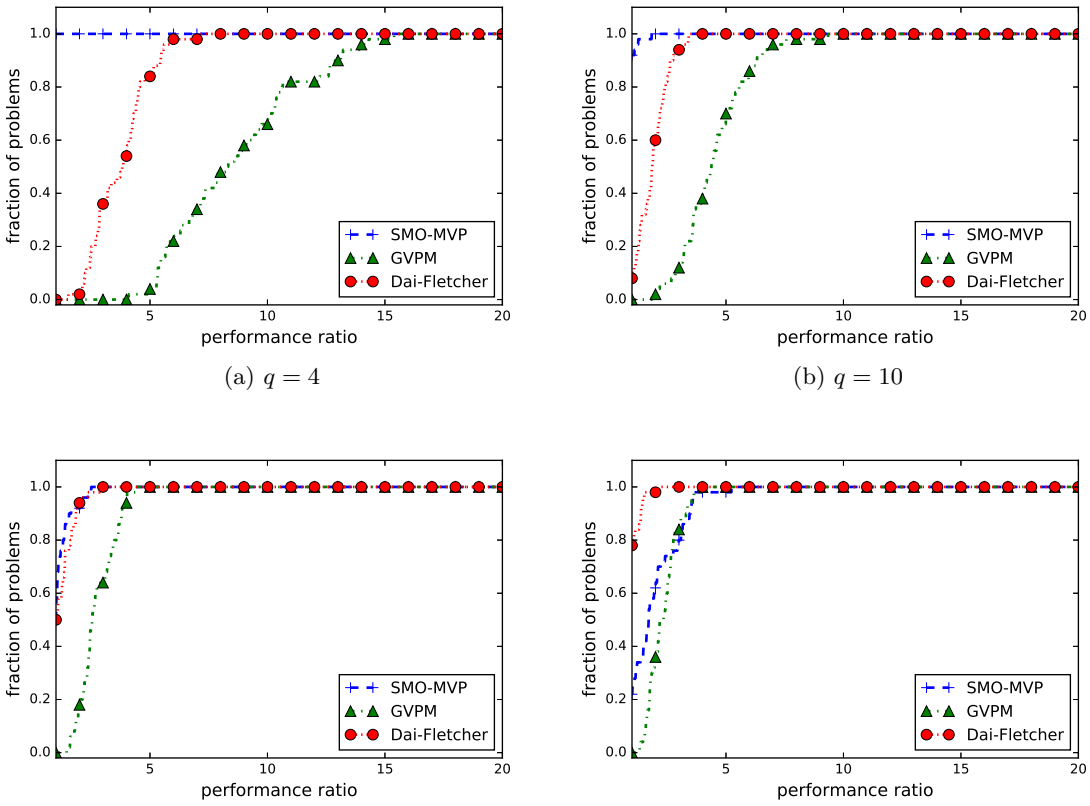
4.3 Computational evidence on inner SMO efficiency

In order to evaluate the efficiency of the inner SMO scheme at solving subproblems arising in Algorithm 1, we compared it against two state-of-art solvers for problems with bounds and a single linear constraint employed in decomposition schemes for SVM training. Namely we considered the Generalized Variable Projection Method (GVPM) proposed by Zanni (2006) and the Dai-Fletcher method described in Dai and Fletcher (2006).

The inner SMO procedure have been implemented from scratch, while we used the implementations of GVPM and Dai-Fletcher methods available at `http://cdm.unimo.it/home/matematica/zanni.luca/`. All three algorithms were then employed in the decomposition scheme whose implementation is highly based on that of LIBSVM (Chang and Lin, 2011). Each of the three solvers employs the warm-star strategy, i.e. they initialize the subproblem variables with the current solution estimate.

The comparison is carried out on 50 Gaussian kernel SVM training problems, generated from data sets `a1a` and `splice` as described in Section 4.1.

For each test problem, we computed the average time spent by each solver to solve one subproblem of q variables during a training run conducted with Algorithm 1 with extended-WSS1. Different values of q have been considered, in particular we repeated the experiment for $q \in \{4, 10, 20, 50\}$. The results are shown in Figure 1. Here, as well as in the rest of the paper, comparisons are carried out by means of performance profiles.



(a) $q = 4$ (b) $q = 10$

(c) $q = 20$ (d) $q = 50$

Figure 1: Performance profiles of mean subproblem solution time required by different solvers (SMO-MVP, GVPM, Dai-Fletcher) in the decomposition method for SVM training with the extended-WSS1, run on 50 RBF kernel SVM training problems from `a1a` and `splice`, for different values of q.

We can see that, for limited values of q, SMO-MVP is almost always the best algorithm, often by a large factor. Dai-Fletcher method appears to be overall superior to GVPM (in accordance with the observations of Zanni (2006)). As the dimension of the working set grows, the gap in performance shrinks: for $q = 20$, the profiles of SMO-MVP and Dai-Fletcher nearly overlap. Finally, when $q = 50$, Dai-Fletcher substantially outperforms the other two methods, which behave similarly.

Now, this trend is in fact not surprising. The good scalability properties of GVPM and Dai-Fletcher algorithms are known from the literature (Zanni, 2006).

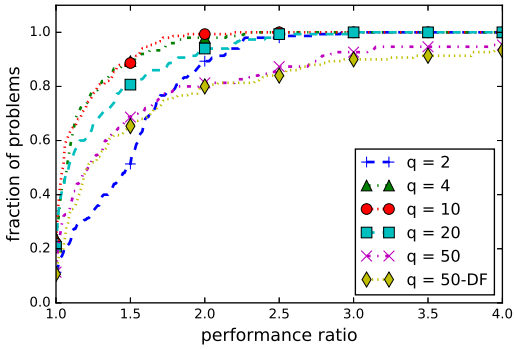
The results of the preceding test make us confident that for General Decomposition Algorithms with $q > 2$ the Two-Level Decomposition scheme is the best one to adopt, at least for values of q not too large.

4.4 Evaluation of different working set sizes

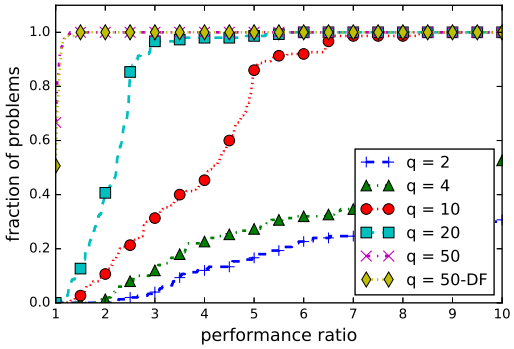
It is now interesting to evaluate the benefits, if any, of using larger working sets, being an efficient subproblems solver available. We thus compare the performance, in terms of runtime, number of iterations and kernel computations of Algorithm 1 equipped with the extended-WSS1 the inner SMO solver, for different values of q. For the case $q = 2$, the employed algorithm is nothing but the standard SMO-MVP on the whole problem. The test for $q = 50$ was repeated by using the Dai-Fletcher solver, as it appears to be, from the results in Figure 1d, the best solver for such working set size.

This experiment is performed on 150 SVM training problems generated from data sets `a1a`, `splice`, `a9a`, `ijcnn1`, `w8a` and `rcv1` as described in Section 4.1. This is a collection of problems from small, medium-sized and large (although not huge) data sets, constituting therefore a diverse enough test suite.

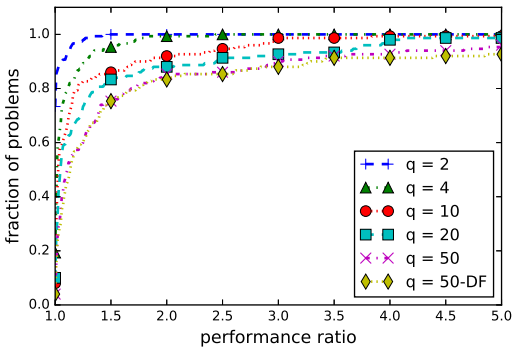
The results of the experiment are shown in Figure 2. Better performance, in terms of runtime, can be observed for the non-extreme working set sizes. As we could reasonably expect, the number of iterations generally decreases as the size of the working set increases. On the other hand, we can see that the number of kernel evaluations has the opposite behavior. We can interpret the result as follows: choosing more variables altogether has the advantage of reducing the number of iterations and consequently the number of times the selection procedure is performed; this is particularly beneficial with the extended-WSS1, since the cost of this selection rule is constant with respect to the WS size; however, the variables are selected in a less careful way: this is paid off by computing possibly unnecessary Hessian columns. Moreover, the computational effort required to solve the subproblem becomes greater as the size of the working set grows. Therefore, intermediate values of q, such as $q = 4$, are not surprisingly good in terms of runtime. It is interesting to note that the performance of $q = 50$ does not change much when the internal solver is changed; we can explain this behavior highlighting that kernel evaluation and working set selection are much more relevant in determining the total runtime w.r.t. the subproblem solution. We can conclude that the availability of an efficient solver for subproblems with more than two variables allows to concretely consider the use of non-SMO decomposition strategy, at least for reasonable values of q. In particular, more sophisticated and combined variables selection rules can be employed, and also the caching mechanism can be exploited in a more systematic way.



(a) Runtime



(b) Number of (outer) iterations



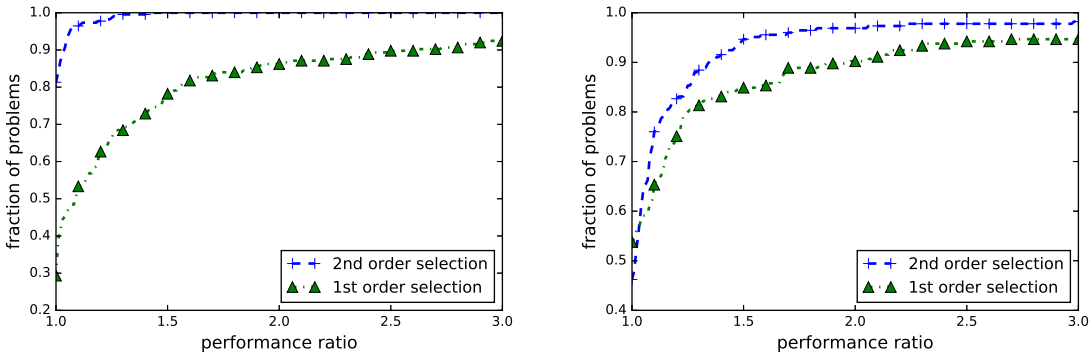
(c) Number of kernel evaluations.

Figure 2: Performance profiles of runtimes (a), iterations (b) and number of kernel evaluations (c) of the decomposition method with extended-WSS1, run on 150 RBF kernel SVM training problems with different values of q. The problems were generated from data sets `a1a`, `splice`, `a9a`, `w8a`, `ijcnn1` and `rcv1`. We employed SMO as inner solver. The test for $q = 50$ was repeated using Dai-Fletcher method since from the results in Figure 1d it appears to be the optimal choice for such working set size.

We now turn to evaluating the performance of the novel working set selection rule.

4.5 Experimental analysis of working set selection rules

We start the discussion on the working set selection rule with a preliminary comparison of WSS1 and WSS2, both in terms of iterations and CPU time. The results are given in Figure 3.



(a) Number of iterations.

(b) Runtime.

Figure 3: Performance profiles comparing the performance, in terms of number of iterations and runtime, of SMO equipped with WSS1 and WSS2, run on 225 RBF kernel SVM training problems. The problems were generated as described in Section 4.1 from data sets `a1a`, `splice`, `leukemia`, `a9a`, `w8a`, `ijcnn1`, `rcv1`, `real-sim` and `covtype`.

We observe that, in terms of iterations, WSS2 is indeed clearly superior to WSS1. This confirms the results shown by Fan et al. (2005), which make WSS2 more appealing especially with large datasets (less iterations are assumed to require less kernel evaluations). However, if we consider the computing time, the difference is less marked and in a good number of cases WSS1 seems a fairly good solution. In particular, in about a half of the situations, WSS1 led to a faster training.

An explanation to this inconsistency between iterations and time can be found in the cost of the working set selection rule itself. In fact, a deeper analysis of the results revealed that this behavior, although common to all problems, is particularly pronounced on problems generated from the small data sets. For such problems the time spent on kernel computations is not only less dominant (even if still relevant) in the composition of total runtime, but also it is almost equal for the two different algorithms, since the Hessian matrix can be entirely stored and thus approximately only Hessian columns associated with support vectors are computed, once and only once. Therefore, the computational cost of the variables selection procedure gets much more weight in such cases.

Finally, let us consider the extensions to the general, non-SMO case of WSS1 and WSS2. The cost of WSS1 procedure does not depend on the number of selected variables; on the other hand, the cost of WSS2 linearly increases with the size of the working set. Therefore, we may expect that the behavior observed in Figure 3 is even more evident when dealing with General Decomposition Algorithms.

Next, we propose a comparison between all combinations of WSS1 and WSS2 for q equal to 2 or 4. In essence, we consider

- 4-WSS-MIX: Algorithm 1 with $q = 4$ and WSS-MIX;
- 2-WSS2: SMO with WSS2;
- 2-WSS1: SMO-MVP;
- 4-WSS1: Algorithm 1 with $q = 4$ and the extended-WSS1, i.e. the rule from Joachims (1999);
- 4-WSS2: Algorithm 1 with $q = 4$ and an extension of WSS2; in particular, the generalization of WSS2 that we considered selects two pairs of variables with WSS2, but for the second pair the "roles" of sets $R(\alpha^k)$ and $S(\alpha^k)$ are inverted, so that it's the variable chosen from $S(\alpha^k)$ the one which is fixed; formally:

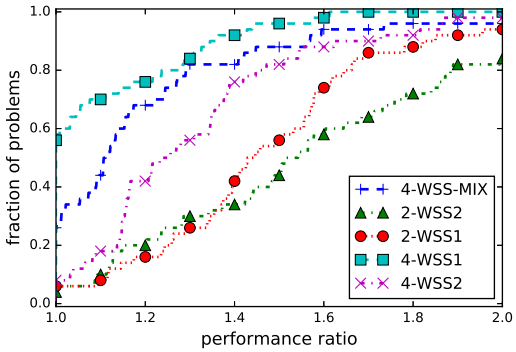
$$\begin{aligned} i_1 &= \arg\max_{h \in R(\alpha^k)} \left\{ -y_h \nabla_h f(\alpha^k) \right\}, \\ j_1 &= \arg\min_{h \in S(\alpha^k)} \left\{ -\frac{\delta_{i_1 h}^2}{p_{i_1 h}} \;\middle|\; -y_h \nabla_h f(\alpha^k) < -y_{i_1} \nabla_{i_1} f(\alpha^k) \right\}, \\ j_2 &= \arg\min_{h \in S(\alpha^k), h \neq j_1} \left\{ -y_h \nabla_h f(\alpha^k) \right\}, \\ i_2 &= \arg\min_{h \in R(\alpha^k), h \neq i_1} \left\{ -\frac{\delta_{j_2 h}^2}{p_{j_2 h}} \;\middle|\; -y_h \nabla_h f(\alpha^k) > -y_{j_2} \nabla_{j_2} f(\alpha^k) \right\}. \end{aligned} \tag{26}$$

We applied the above five algorithms to 175 Gaussian kernel SVM training problems, obtained from data sets `a1a`, `splice`, `a9a`, `w8a`, `ijcnn1`, `rcv1` and `real-sim` as described in Section 4.1. Concerning the methods with $q = 4$, we employed the TLD-ISMO scheme.

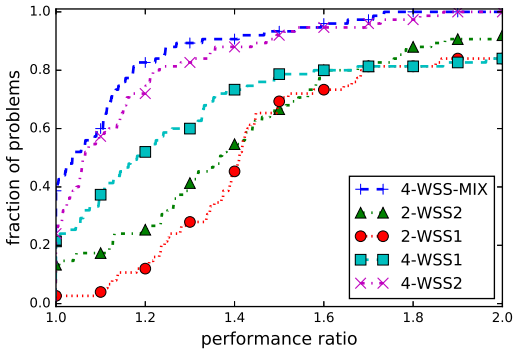
The comparison has been made on the basis of runtime, number of iterations and number of kernel evaluations. These three quantities should provide a sufficiently good insight about the behavior of the algorithms. We report in Figures 4 and 5 the results of the tests. Figures 4a, 4c and 4e illustrate the results on the problems from small data sets (`a1a`, `splice`); Figures 4b, 4d and 4f concern the medium-sized data sets (`a9a`, `w8a`, `ijcnn1`); Figures 5a, 5c and 5e show the results on the problems from large data sets (`rcv1`, `real-sim`); finally, Figures 5b, 5d and 5f summarize the results upon all 175 problems.

We can observe that with small data sets choosing larger working sets is convenient in terms of runtime, in particular by using WSS1. In fact, doubling the size of the working set on average reduces the number of iterations to slightly less than a half. As previously outlined, the number of kernel evaluations is almost constant w.r.t. the WSSRs. WSS2 and WSS-MIX are slightly more efficient, in terms of number of iterations, w.r.t. WSS1; however, when it comes to runtime, WSS1 seems to be the best choice; we can thus deduce that the cost of the selection procedure itself is particularly relevant with these problems.

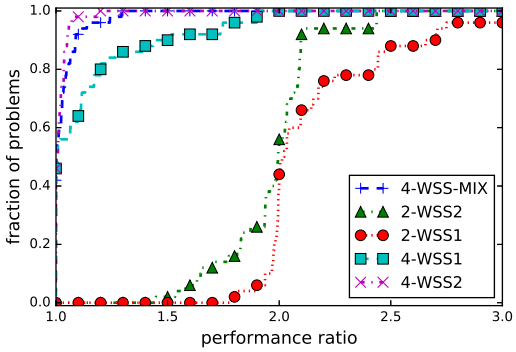
The situation is quite different with large data sets. The presence of at least one pair of variables selected by WSS2 is often useful to reduce the number of iterations. This quantity is indeed strictly related to the number of kernel evaluations in this case, since the cache



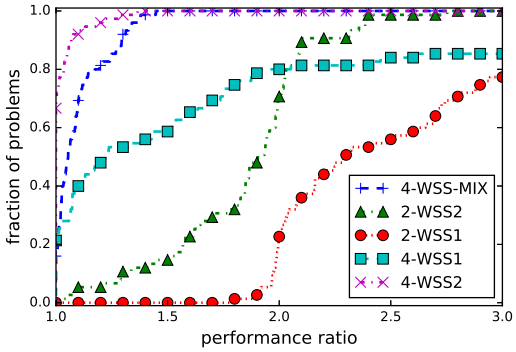
(a) small data sets - runtime



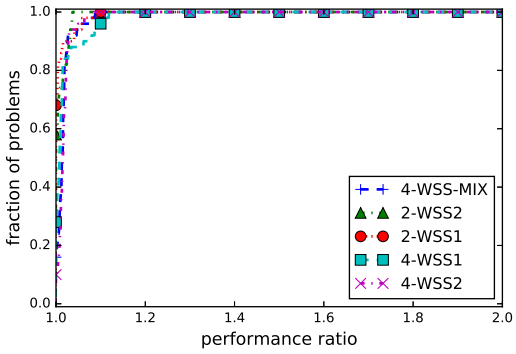
(b) medium-sized data sets - runtime



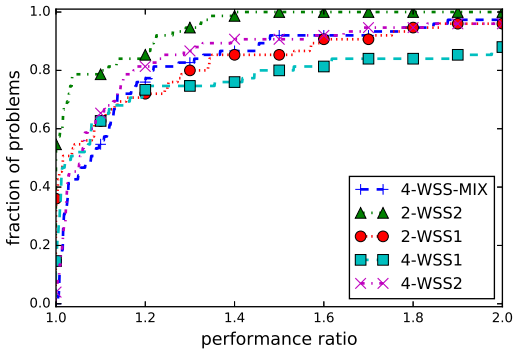
(c) small data sets - number of (outer) iterations



(d) medium data sets - number of (outer) iterations

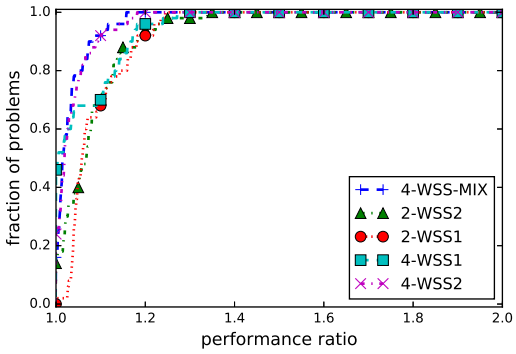


(e) small data sets - number of kernel evaluations

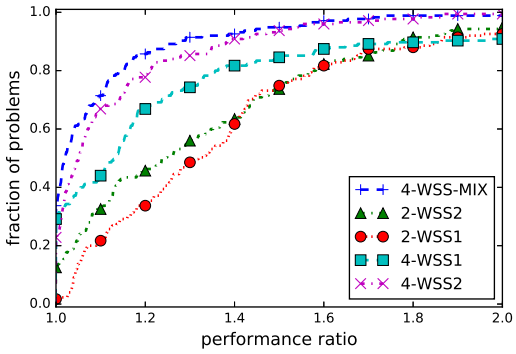


(f) medium data sets - number of kernel evaluations

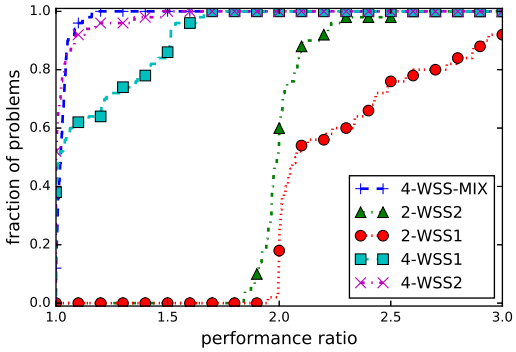
Figure 4: Performance profiles comparing the performance, in terms of runtime (first row), number of (outer) iterations (second row) and number of kernel columns evaluations (third row), of Algorithm 1 equipped with combinations of WSS1 and WSS2, run on problems generated from the small data sets `a1a`, `splice` (first column) and from the medium-sized data sets `a9a`, `w8a`, `ijcnn1` (second column).



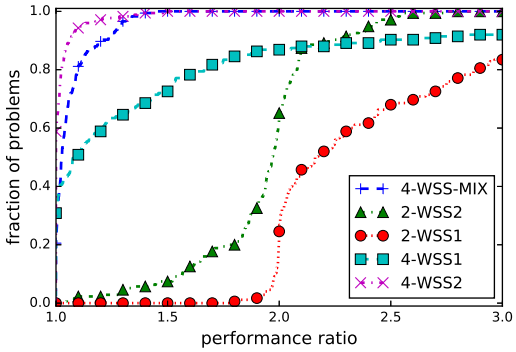
(a) large data sets - runtime



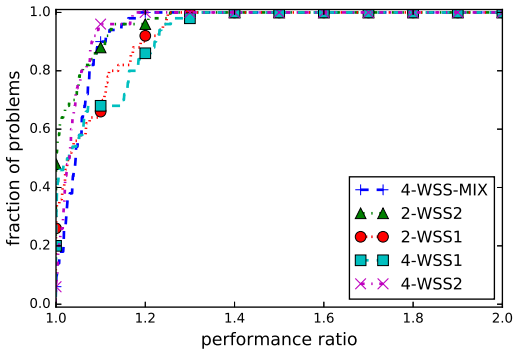
(b) all data sets - runtime



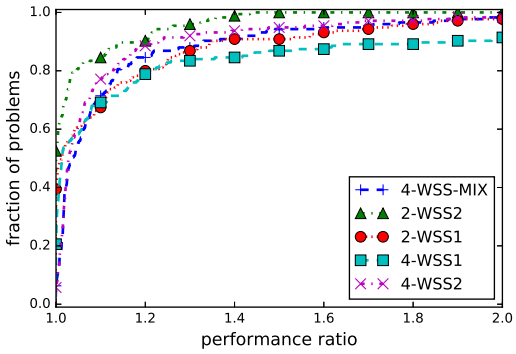
(c) large data sets - number of (outer) iterations



(d) all data sets - number of (outer) iterations



(e) large data sets - number of kernel evaluations



(f) all data sets - number of kernel evaluations

Figure 5: Performance profiles comparing the performance, in terms of runtime (first row), number of (outer) iterations (second row) and number of kernel columns evaluations (third row), of Algorithm 1 equipped with combinations of WSS1 and WSS2, run on problems generated from the large data sets `rcv1`, `real-sim` (first column) and from all the data sets `a1a`, `splice`, `a9a`, `w8a`, `ijcnn1`, `rcv1`, `real-sim`.

is relatively small. Being the runtime almost entirely determined by the cost of kernel computations, using second order information is thus beneficial.

The case of medium-sized data sets appears to be the most complicated. The number of iterations, the cost of working set selection rule and the number of kernel evaluations are all relevant to the total runtime. WSS-MIX apparently balances these components of the cost to obtain the best performance overall. It is worth mentioning that for some of these problems, the absence of variables selected by second order information resulted in particularly critical cases in terms of iterations.

The aggregate results from all 175 problems show a behavior, not surprisingly, somewhere in between the cases of large data sets and small data sets and similar to the case of medium-sized data sets. One variable selected using second order information is necessary in order to avoid disasters in terms of iterations; at the same time, selecting more variables at once, especially if most of them are chosen by the cheap WSS1, can reduce the time spent on the selection procedures. WSS-MIX thus appears as the most suited option to be applied on a generic problem.

4.6 Analysis of the effects of adding cached variables

We conclude the analysis on the working set selection rule, before evaluating the efficiency of the full method, by analyzing the effect of adding additional cached variables to the working set.

We therefore analyze how the selection of additional "cached" variables actually affects the total number of kernel evaluations required by Algorithm 1 on both cases of small and large problems.

With small data sets, the Hessian matrix can entirely be stored; therefore, Hessian columns corresponding to support vectors need to be computed only once, while, with reasonable WSS rules, few other columns are computed throughout the optimization process. This consideration is supported by empirical evidence, as shown in Figure 6a, concerning the number of kernels evaluated by Algorithm 1 with different WSS rules.

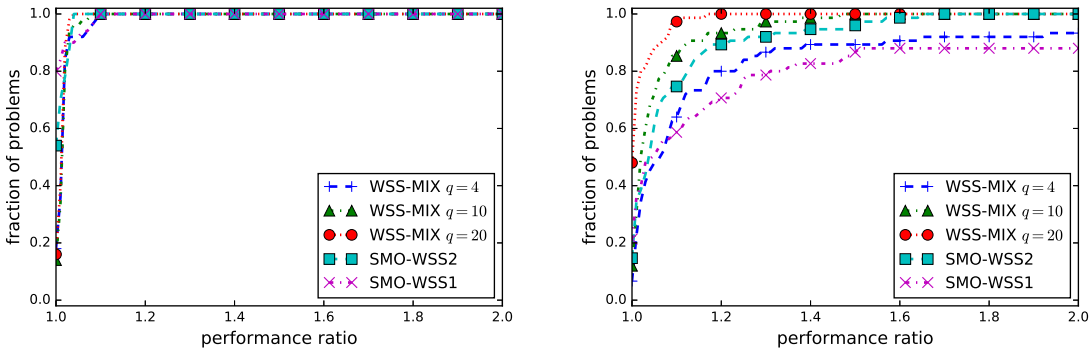
On the contrary, with large problems, the selection of the extra variables corresponding to cached Hessian columns helps much at speeding up the entire process. In fact, as we see in Figure 6b, base WSS-MIX, being an hybrid rule combining WSS1 and WSS2, has an intermediate behavior w.r.t. these two basic rules. When the working set is enlarged with the addition of the stored variables, however, the performance greatly improves.

4.7 Overall evaluation of TLD-ISMO with WSS-MIX

In this section, the overall efficiency performance of Algorithm 3 is finally evaluated. We compare the TLD-ISMO to the most widely used decomposition schemes for SVM, i.e. SMO methods equipped with WSS1 and WSS2.

Concerning the working set selection rule of TLD-ISMO, rule (21) was integrated with the selection of a number of variables from the preceding working according to Algorithm 5. The final dimension of the working set for each problem was decided by means of the heuristic rule (22).

Since we solve a convex problem and for all algorithms we used the same stopping condition, we obtain from the different algorithms equivalent solutions, except for numerical



(a) 50 small problems

(b) 75 large problems

Figure 6: Performance profiles in number of Hessian columns computed by Algorithm 1 with different WSS rules. WSS-MIX with $q = 10$ and $q = 20$ include the addition of 6 and 16 "cached" variables respectively. Figure 6a concerns 50 test RBF kernel SVM problems generated from small data sets `a1a` and `splice` as described in Section 4.1. Figure 6b describes the results of the experiments on 75 problems from large data sets `rcv1`, `real-sim` and the very large `covtype`.

imprecision issues. This is mirrored also in the test accuracy of the models obtained by the different algorithms. Therefore, it makes sense comparing the algorithms in terms of efficiency, i.e. runtime. We first ran the three algorithms on the 225 RBF kernel SVM training problems generated from data sets `a1a`, `splice`, `leukemia`, `a9a`, `w8a`, `ijcnn1`, `rcv1`, `real-sim` and `covtype` as described in Section 4.1.

A summary of the obtained results can be found in Figures 7 and 8. In Figure 7 the performance profile of the efficiency of the three algorithms on all 225 problems is shown. We can see that not only TLD-ISMO in general performs better than SMO algorithms, but also it often provides a significant speed-up.

In Figure 8 we show the same comparison, but we focus the analysis on the different data sets scales. We can observe that the dominance of TLD-ISMO spreads to any situation.

We repeated the experiment on 8 extra problems, generated as described in Section 4.1 from data sets `phishing`, `cifar-resnetv2-emb` and `news20`, `cod-rna`, `skin_nonskin` and (subsets of) `epsilon`, `SUSY` and `HIGGS`. For the tests on the latter two data sets we introduced a time limit of 100,000 seconds per single run, since some hyperparameter configurations led to problems that proved to be too hard for the decomposition algorithms with the hardware at our disposal. This test was carried out so as to ensure that the efficiency of TLD-ISMO portrayed by Figures 7 and 8 was not the result of an excessively specific implementation.

In fact, the results observable in Figures 9 and 10 suggest that it is not the case: on the contrary, the advantage of TLD-ISMO is even more pronounced.

Performance profiles are a useful tool to summarize and explain the behavior of several algorithms on an enormous number of test problems. To enrich the analysis, we provide some complementary information about runtimes absolute values. In the model validation

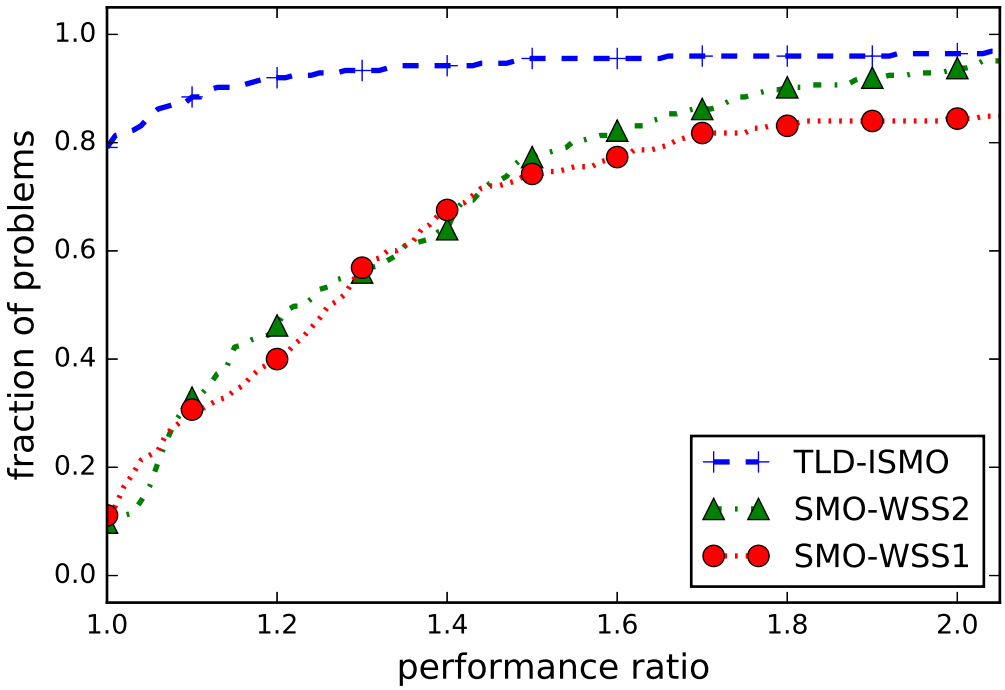
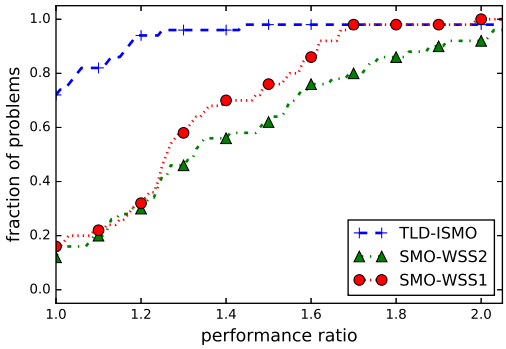
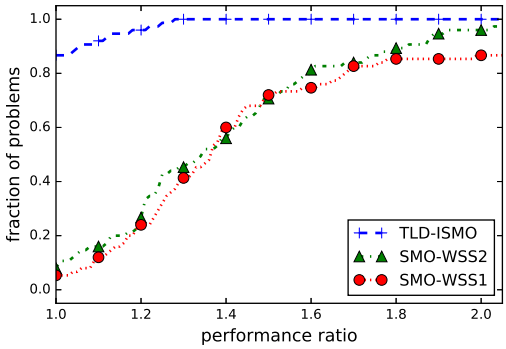


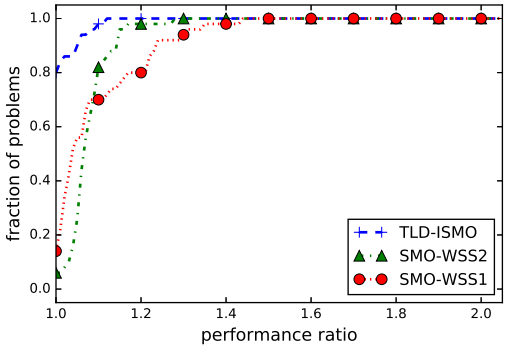
Figure 7: Comparison of the efficiency of TLD-ISMO and SMO algorithms (equipped with WSS1 and WSS2) on all 225 test problems.



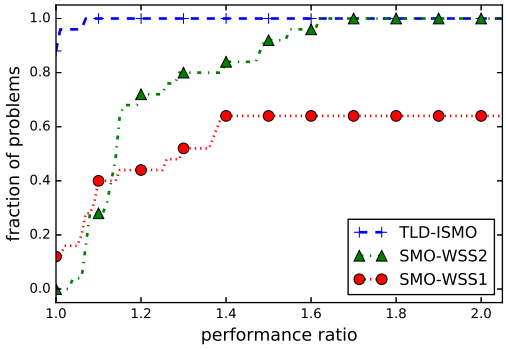
(a) 75 problems from the three small data sets.



(b) 75 problems from the three medium-sized data sets.

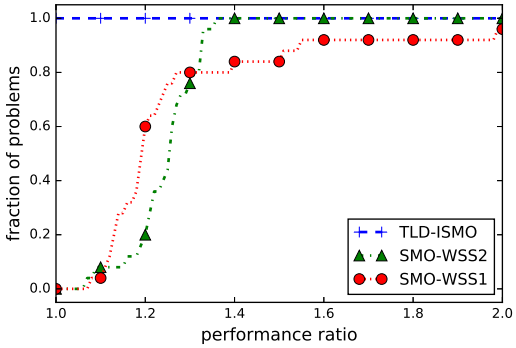


(c) 50 problems from the two large data sets.

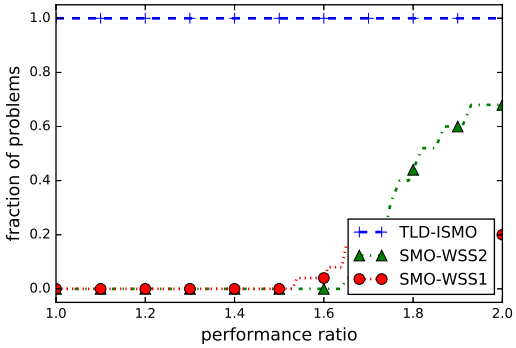


(d) 25 problems from the huge data set.

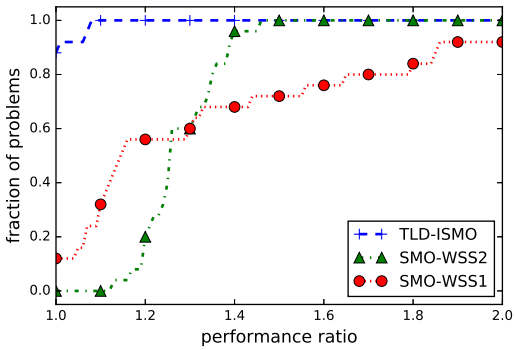
Figure 8: Performance profiles comparing the efficiency, in terms of runtime, of TLD-ISMO and SMO (equipped with WSS1 and WSS2), on different classes of RBF kernel SVM problems. The problems are obtained, by varying the hyperparameters, from data sets `a1a`, `splice`, `leukemia` (small), `a9a`, `w8a`, `ijcnn1` (medium-sized), `rcv1`, `real-sim` (large) and `covtype` (huge).



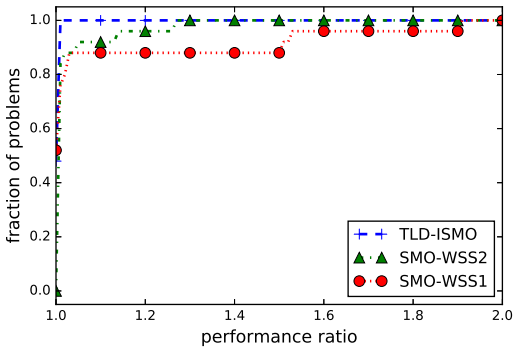
(a) 25 problems from `phishing` data set.



(b) 25 problems from `cod-rna` data set.

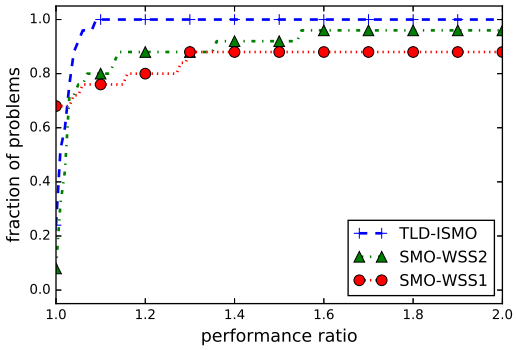


(c) 25 problems from `skin_nonskin` data set.

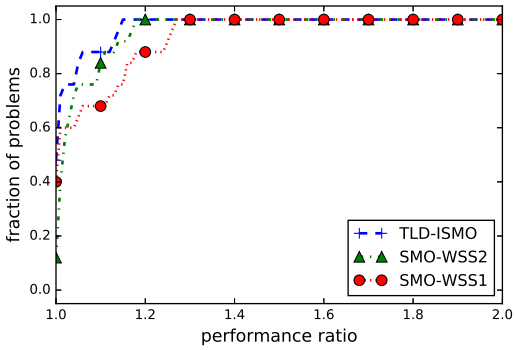


(d) 25 problems from a subset of `epsilon` data set.

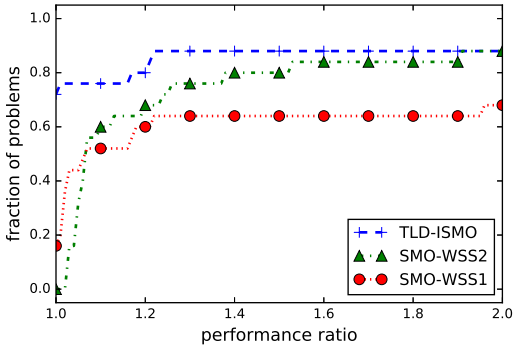
Figure 9: Performance profiles comparing the efficiency, in terms of runtime, of TLD-ISMO and SMO (equipped with WSS1 and WSS2), on RBF kernel SVM problems problems from four different data sets: `phishing`, `cod-rna`, `skin_nonskin` and a subset of `epsilon` .



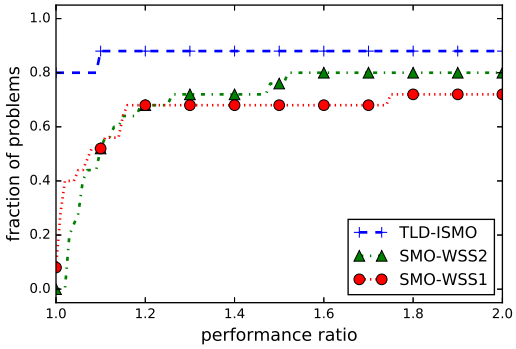
(a) 25 problems from `cifar-resnetv2-emb` data set.



(b) 25 problems from `news20` data set.



(c) 25 problems from a subset of `SUSY` data set.



(d) 25 problems from from a subset of `HIGGS` data set.

Figure 10: Performance profiles comparing the efficiency, in terms of runtime, of TLD-ISMO and SMO (equipped with WSS1 and WSS2), on RBF kernel SVM problems problems from four different data sets: `cifar-resnetv2-emb`, `news20` and subsets of `SUSY` and `HIGGS`.

dataset	TLD-ISMO	SMO-WSS1	SMO-WSS2
a1a	**2.20**	2.79	2.64
splice	**1.43**	1.68	1.78
leukemia	**0.51**	0.52	0.56
w8a	**1808.37**	2213.93	2275.07
ijcnn1	**2023.05**	5109.01	2541.04
a9a	**3003.09**	7708.22	4203.10
rcv1	**2432.89**	2585.81	2509.33
real-sim	**22780.93**	24116.95	24495.85
covtype	**518847.84**	2213224.58	651239.04
epsilon-subset	**182127.99**	217016.02	191457.61
phishing	**64.64**	81.12	80.64
cod-rna	**1999.17**	26490.30	4698.16
skin_nonskin	**10465.35**	12810.54	13364.90
cifar-resnetv2-emb	**25876.53**	35630.19	29719.72
news20	**10704.85**	11263.44	10980.03
SUSY-subset	**715706 (3)**	918867 (5)	746725 (3)
HIGGS-subset	**1011667 (3)**	1166775 (5)	1119047 (5)

Table 3: Total time (in seconds) spent by TLD-ISMO, SMO-WSS1 and SMO-WSS2 at solving the problem grid generated as described in Section 4.1 by each dataset. The number in brackets, when present, denotes the number of times the solver could not complete the optimization process within the time limit of 100.000 seconds.

phase of machine learning applications, which is often the most time consuming, users are typically required to solve an entire grid of problems. So, great interest lies in the sum of runtimes obtained over such grids.

For this reason, we show in Table 3 the total runtime for the three considered algorithms over each grid of problems generated from data sets of Tables 1 and 2. We can see that TLD-ISMO has been faster at solving the 25 problems with respect to the two versions of SMO with each of the 17 employed data sets. In particular, in some cases, e.g. for `a9a`, `covtype` and `cod-rna`, the amount of time saved is massive. Also, in the case of `HIGGS`, in addition to reducing the total runtime, TLD-ISMO proved to be able to complete the training process within the time limit in cases where the SMO procedures could not.

4.8 Relevance of the cache size

Up to this point we have been keeping the cache size fixed throughout all of our experiments. The rational behind such a choice is that of analyzing cases where only a certain percentage of the Hessian matrix can be stored, no matter of the actual size of the data.

However, it is indeed interesting seeing what the real impact is of exploiting, when possible, a larger amount of memory. We conducted supplementary experiments to this aim.

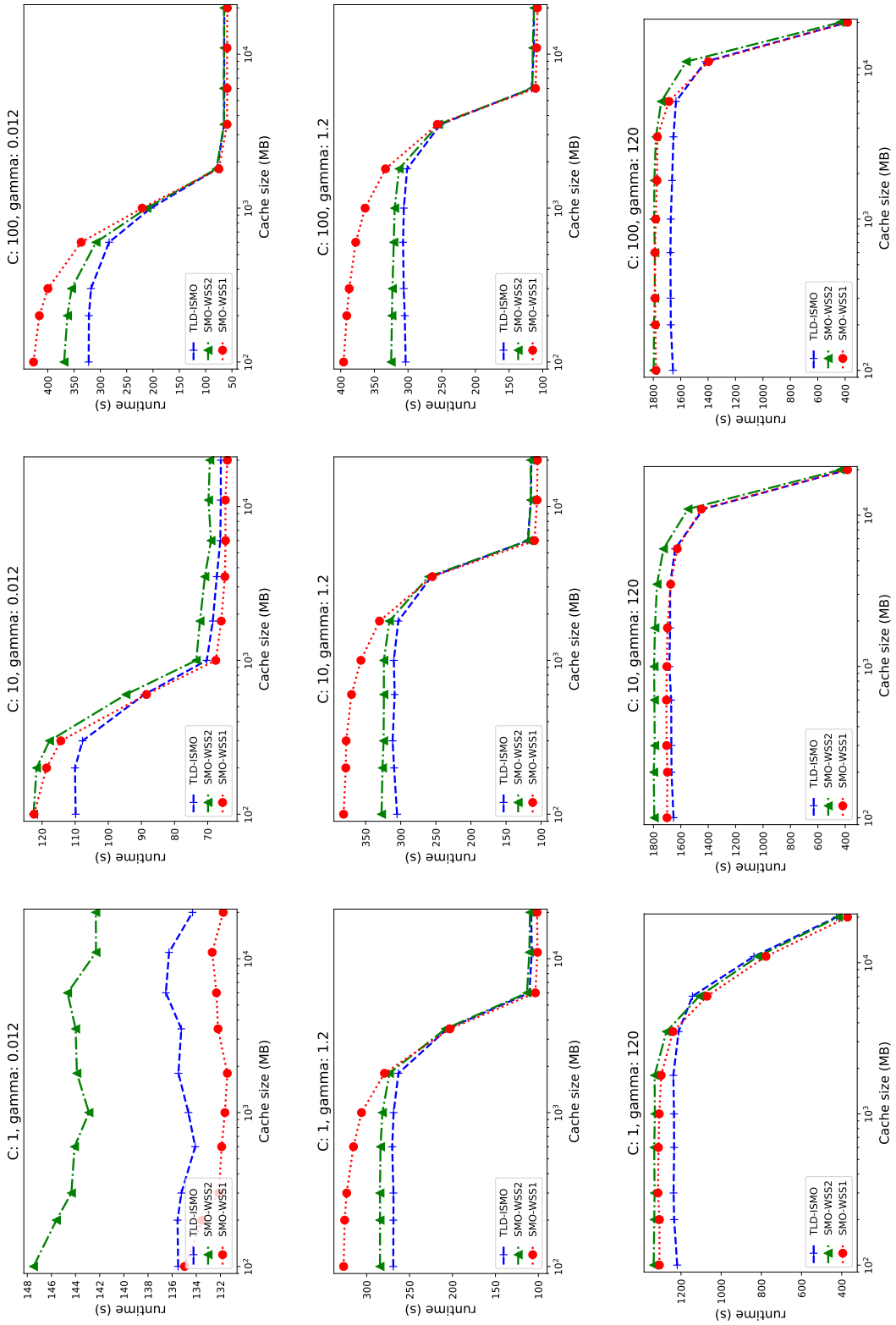


Figure 11: Runtimes obtained by TLD-ISMO and SMO (equipped with WSS1 and WSS2) on different problems, varying the size of the cache. The problems are obtained from the `real-sim` data set, with different hyperparameters settings.

For this experiment we employed a different computer w.r.t. the tests described in the previous sections; specifically, we used an Intel Xeon Gold 5120 with 26 cores and a 40 GB RAM. We made this choice in order to have enough memory for possibly storing the entire Hessian matrix of one of the large problems, which are particularly significant in practice. We indeed picked the `real-sim` data set, i.e., the largest data set in our benchmark whose Hessian matrix is less than 40 GB (it actually occupies nearly 20 GB), to carry out the experiment.

The experiment consisted in repeating the training process on a set of SVM training problems, with a number of different setups in terms of cache size, using TLD-ISMO ($q = 20$) and SMO equipped with WSS1 and WSS2 . Since we had to vary the cache size parameter in addition to the hyperparameters C and γ, this time we defined a 3×3 grid of problems, generated from the `real-sim` data set. We tested for each problem 10 different values for the cache size, selected by a logarithmic scale from the base size 100 MB to the 20 GB required to store the entire Hessian matrix.

The results, in terms of runtime, are shown in Figure 11. We can observe that in most cases the performance of the algorithms does not have significant improvements as the cache size increases. This trend only changes when the available memory starts to be relatively large w.r.t. the full Hessian size. We can also see that TLD-ISMO typically maintains its competitiveness, independently of the cache size.

In order to better understand these results, analyzing how the number of kernel computations changes with the cache size is interesting. Indeed, kernel computation is the component of the computational cost that can be reduced by increasing the memory at one's disposal. We show in Figure 12 how many times the Hessian columns have been (re)computed in each run of each considered algorithm. We also plot lines that indicate the number of columns of the entire Hessian matrix, i.e., the number of columns that are computed if kernels were precomputed and stored, and of the number of support vectors, i.e., approximately the number of the columns that are actually used during the optimization process.

We can then see that the plots resemble those concerning the runtimes, particularly for problems with larger C, which is not surprising. We can also note that the number of computed columns really always converges, as the cache size increases, to approximately the number of support vectors. Moreover, we see that in many cases the decomposition methods compute, if enough memory is available, less kernels than we would have done by pre-computing and storing the entire Hessian matrix.

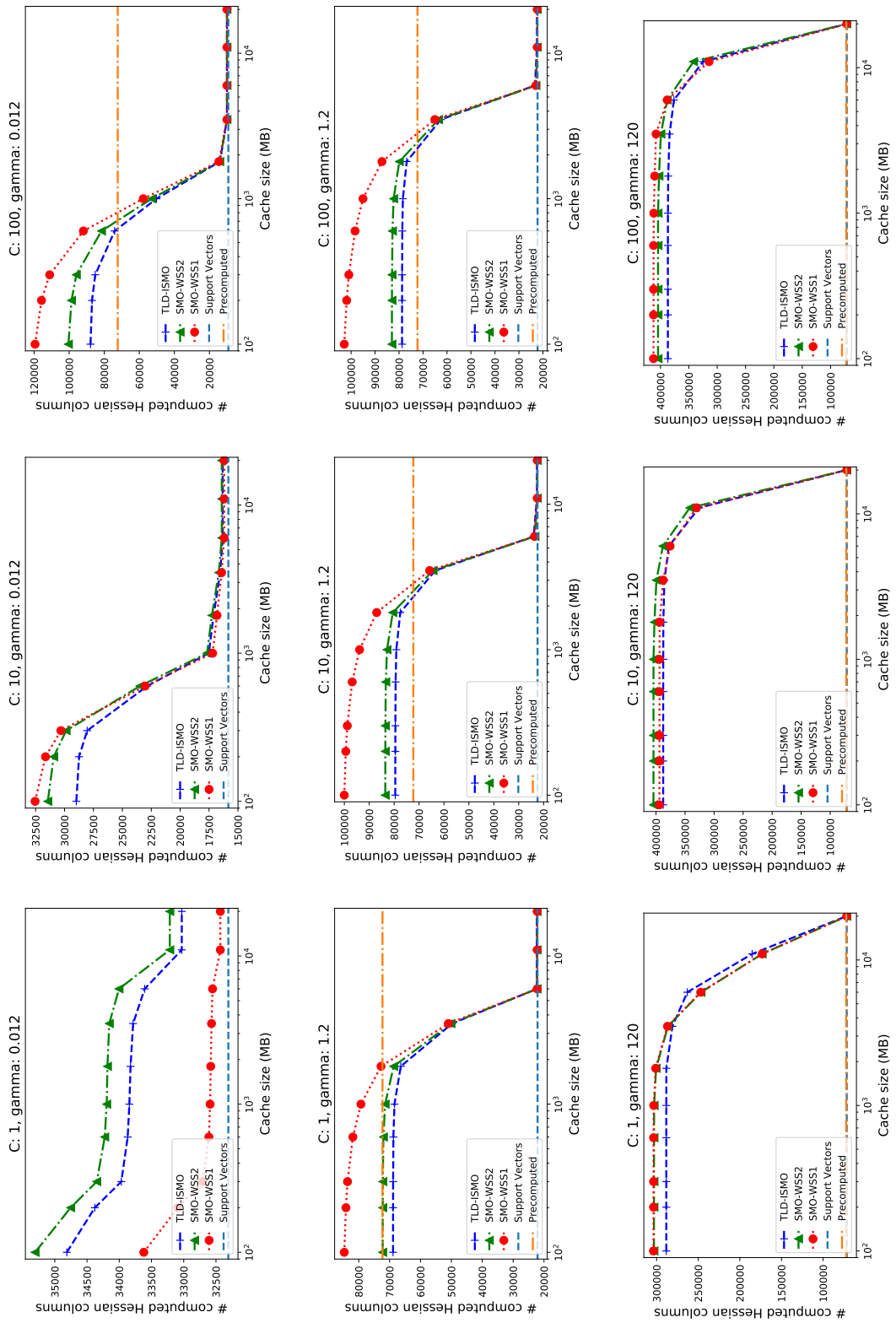


Figure 12: Number of Hessian columns (re)computed by TLD-ISMO and SMO (equipped with WSS1 and WSS2) on different problems, varying the size of the cache. The problems are obtained from the `real-sim` data set, with different hyperparameters settings.

5. Conclusions

Decomposition schemes are a popular and effective way to solve the optimization problem underlying the training stage of kernel SVMs. A decomposition algorithm reduces the training problem to the solution of several smaller sub-problems of 2 or more variables. However, solving sub-problems of more than two variables is often significantly harder and, thus, not computationally convenient.

In this work we proposed a novel way to deal with sub-problems of more than two variables. Namely, since the sub-problems share the same structure and formulation of the original SVM training problem, we solved the sub-problems with the same SMO technique that is used in state-of-art solvers like LIBSVM to solve the original problem.

Numerical evidence is provided to show that employing SMO to solve the sub-problems compares favorably, for up to 50 variables, with other state-of-the-art solvers designed to solve sub-problems of more than two variables, like GVPM and the Dai-Fletcher method.

We then equipped the decomposition algorithm with a novel working set selection rule which exploits both first and second order information, with the addition of "cached variables" for large data sets. The rule yields sub-problems of a minimum of 4 variables for small data sets to a few tens for larger ones.

Finally we compared the whole decomposition scheme, where the sub-problems generated by the novel working set selection rule are solved with SMO, against state-of-art solvers like LIBSVM. The numerical evidence shows that the proposed approach significantly outperforms LIBSVM on a large and diverse range of data sets.

Future work will be devoted to investigate the use of sub-problems of more than 50 variables, where the benefits of the current approach start to wear off. A potentially effective way to deal with larger sub-problems could be that of further decomposing them into sub-sub-problem and use, again, SMO to solve them in a recursive way.

Moreover, an in-depth study to investigate the effects of the shrinking heuristic combined with a working set selection rule exploiting cached information, such as the one proposed in this work, might be of particular interest.

Acknowledgments

We would like to thank the action editor and the anoymous reviewers whose comments helped us to improve the quality of the paper.

References

Shigeo Abe. Batch support vector training based on exact incremental training. In *International Conference on Artificial Neural Networks*, pages 295–304. Springer, 2008.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152. ACM, 1992.

Leon Bottou, Olivier Chapelle, Dennis DeCoste, and Jason Weston. *Support Vector Machine Solvers*. MIT Press, 2007.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

Chih-Chung Chang, Chih-Wei Hsu, and Chih-Jen Lin. The analysis of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 11(4):1003–1008, 2000.

Pai-Hsuen Chen, Rong-En Fan, and Chih-Jen Lin. A study on SMO-type decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 17(4): 893–908, 2006.

Ronan Collobert, Samy Bengio, and Yoshua Bengio. A parallel mixture of SVMs for very large scale problems. In *Advances in Neural Information Processing Systems*, pages 633–640, 2002.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3): 273–297, 1995.

Yu-Hong Dai and Roger Fletcher. New algorithms for singly linearly constrained quadratic programs subject to lower and upper bounds. *Mathematical Programming*, 106(3):403–421, 2006.

Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

Dheeru Dua and Efi Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Rong-En Fan, Pai-Hsuen Chen, and Chih-Jen Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6(Dec):1889–1918, 2005.

Thilo-Thomas Frie, Nello Cristianini, and Colin Campbell. The kernel-adatron algorithm: a fast and simple learning procedure for support vector machines. In *Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98)*, pages 188–196, 1998.

Tobias Glasmachers and Christian Igel. Maximum-gain working set selection for SVMs. *Journal of Machine Learning Research*, 7(Jul):1437–1466, 2006.

Todd R. Golub, Donna K. Slonim, Pablo Tamayo, Christine Huard, Michelle Gaasenbeek, Jill P. Mesirov, Hilary Coller, Mignon L. Loh, James R. Downing, Mark A. Caligiuri, et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. *Science*, 286(5439):531–537, 1999.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

Chih-Wei Hsu and Chih-Jen Lin. A simple decomposition method for support vector machines. *Machine Learning*, 46(1-3):291–314, 2002.

Thorsten Joachims. Making large-scale support vector machine learning practical. In Bernhard Schölkopf, Christopher J. C. Burges, and Alexander J. Smola, editors, *Advances in Kernel Methods*, pages 169–184. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-19416-3.

Sathiya S. Keerthi and Dennis DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6(Mar):341–361, 2005.

Sathiya S. Keerthi, Shirish K. Shevade, Chiranjib Bhattacharyya, and Krishna R.K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE Transactions on Neural Networks*, 11(1):124–136, 2000.

Sathiya S. Keerthi, Shirish K. Shevade, Chiranjib Bhattacharyya, and Karuturi R.K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, 2001.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5(Apr):361–397, 2004.

Chih-Jen Lin. On the convergence of the decomposition method for support vector machines. *IEEE Transactions on Neural Networks*, 12(6):1288–1298, 2001.

Chih-Jen Lin. Asymptotic convergence of an SMO algorithm without any assumptions. *IEEE Transactions on Neural networks*, 13(1):248–250, 2002a.

Chih-Jen Lin. A formal analysis of stopping criteria of decomposition methods for support vector machines. *IEEE Transactions on Neural Networks*, 13(5):1045–1052, 2002b.

Chih-Jen Lin and Jorge J Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

Chih-Jen Lin, Stefano Lucidi, Laura Palagi, Arnaldo Risi, and Marco Sciandrone. Decomposition algorithm model for singly linearly-constrained problems subject to lower and upper bounds. *Journal of Optimization Theory and Applications*, 141(1):107–126, 2009.

Stefano Lucidi, Laura Palagi, Arnaldo Risi, and Marco Sciandrone. A convergent decomposition algorithm for support vector machines. *Computational Optimization and Applications*, 38(2):217–234, 2007.

Stefano Lucidi, Laura Palagi, Arnaldo Risi, and Marco Sciandrone. A convergent hybrid decomposition algorithm model for SVM training. *IEEE Transactions on Neural Networks*, 20(6):1055–1060, 2009.

Olvi L Mangasarian and David R Musicant. Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10(5):1032–1037, 1999.

Edgar Osuna, Robert Freund, and Federico Girosi. An improved training algorithm for support vector machines. In *Neural Networks for Signal Processing VII. Proceedings of the 1997 IEEE Signal Processing Society Workshop*, pages 276–285. IEEE, 1997.

Fernando Pérez-Cruz, Anıbal R Figueiras-Vidal, and Antonio Artés-Rodríguez. Double chunking for solving svms for very large datasets. *Proceedings of Learning*, 2004.

Veronica Piccialli and Marco Sciandrone. Nonlinear optimization and support vector machines. *4OR*, 16(2):111–149, 2018.

John C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

Danil Prokhorov. IJCNN 2001 neural network competition. *Slide presentation in IJCNN*, 2001.

Craig Saunders, Mark O Stitson, Jason Weston, Leon Bottou, A Smola, et al. Support vector machine reference manual. *Technical Report CSD-TR-98-03*, 1998.

Thomas Serafini and Luca Zanni. On the working set selection in gradient projection-based decomposition techniques for support vector machines. *Optimization Methods and Software*, 20(4-5):583–596, 2005.

Norikazu Takahashi and Tetsuo Nishi. Global convergence of decomposition learning methods for support vector machines. *IEEE Transactions on Neural Networks*, 17(6):1362–1369, 2006.

Andrew V Uzilov, Joshua M Keegan, and David H Mathews. Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC Bioinformatics*, 7(1):173, 2006.

Robert J Vanderbei. Loqo: An interior point code for quadratic programming. *Optimization Methods and Software*, 11(1-4):451–484, 1999.

SVM Vishwanathan and M Narasimha Murty. SSVM: a simple SVM algorithm. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02.*, volume 3, pages 2393–2398. IEEE, 2002.

Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1-2):41–75, 2011.

Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. An improved GLMNET for l1-regularized logistic regression. *Journal of Machine Learning Research*, 13(Jun):1999–2030, 2012.

Luca Zanni. An improved gradient projection-based decomposition technique for support vector machines. *Computational Management Science*, 3(2):131–145, 2006.

Qiaozhi Zhang, Di Wang, and Yanguo Wang. Convergence of decomposition methods for support vector machines. *Neurocomputing*, 317:179–187, 2018.